

BYTE

LISTINGS
SUPPLEMENT

July–September 1986



INDEX

July

ANAGRAM	PAS....5	FORMULA	MOD...168	SETPROB	PAS....8	MACAPP	USE...258
ASCSTR	ASM...216	INTSGN	ASM...85	SETSTR	ASM...79	MAPPER	BAS...261
BROCHURE	DOC...60	LENSTR	ASM...87	SIZFIL	ASM...98	MICROS	DAT...265
CALC	MOD...127	LISPTST	DOC...99	SIZSTR	ASM...90	NORMAL	SRC...253
CELL	DEF...181	LISTING1	...201	SPREAD	DEF...159	PALIND	BAS...268
CELL1	MOD...182	LISTING2	...202	SPREAD1	MOD...161	PALIND	RUN...269
CELL2	MOD...173	LOCSTR	ASM...93	SPREAD2	MOD...146	POWSER	BAS...269
CELLLIST	DEF...169	LOGPSL	ASM...83	SPREAD3	MOD...136	POWSER	RUN...270
CELLLIST	MOD...179	LUG	INP...48	SRCHFOR	PAS...127	READFORT	ME...271
CHARSTUF	DEF...157	LWCSTR	ASM...88	STRINGST	DEF...160	TYPES	DOC...271
CHARSTUF	MOD...159	MEMAVA	ASM...58	STRINGST	MOD...186		
CHKDUP	ASM...10	MISC	DEF...185	TEST	SEQ...217		
CHOPWR	ASM...59	MISC	MOD...185	TESTDC	SEQ...217		
COMMANDP	DEF...179	MODSTR	ASM...89	TIME	ASM...80		
COMMANDP	MOD...170	MOVSTR	ASM...56	TRUSS2	BAS...218		
CONFRM	ASM...82	MSP	PAS...103	UGHBUG	ASM...232		
CONSTR	ASM...86	MYSTOR1	DEF...164	UPCSTR	ASM...12		
CRITIBM	BAS...211	MYSTOR1	MOD...165	VISC	BAS...188		
CRITICAL	BAS...197	MYSTOR2	DEF...166	VISCIBM	BAS...202		
DATE	ASM...12	MYSTOR2	MOD...166	WTRANK	PAS...128		
DEFDRV	ASM...96	MYTERMIN	DEF...146				
DELFIL	ASM...92	NUMTOSTR	DEF...163				
DISKIO	DEF...167	NUMTOSTR	MOD...164				
DISKIO	MOD...130	PACER	ASM...83				
DISPLAYH	DEF...140	PAKSTR	ASM...81				
DISPLAYH	MOD...150	PCRACK	BAS...102				
DSKFRE	ASM...97	PUNCH	INP...47				
DSKSPC	ASM...55	READ-ME	MSP...127				
ENDSTR	ASM...91	READTHIS	...135				
EVALUAT	DEF...140	RESSTR	ASM...95				
EVALUAT	MOD...140	SAFE2SOL	FOR...28				
EXPSTR	ASM...46	SAFE2SUB	FOR...13				
FILSTR	ASM...89	SCREENHA	DEF...165				
FORMULA	DEF...168	SCREENHA	MOD...154				

August

CITIES	DAT...265
CUBESUM	RUN...268
CUBESUM	BAS...267
CYTOC	DAT...265
DENSITY	SRC...253
FACTSUM	BAS...266
FACTSUM	RUN...267
LETS	SRC...254

September

BREAKPT	ASM...277
BRKPTCOM	BAS...278
CCITT	C...279
EXPS	ST1...279
SDLC	ASM...286
SKAM	BAS...288
SKAM1	BAS...291
TBPROLOG	TST...294
XMODEM	ASM...299
XMODEM	C...300

BYTE

SENIOR VICE PRESIDENT/PUBLISHER
HARRY L. BROWN
EDITOR IN CHIEF
PHILIP LEMMONS

BIX

MANAGING EDITOR, BYTE
FREDERIC S. LANGA

ASSISTANT MANAGING EDITOR

GLENN HARTWIG

CONSULTING EDITORS

STEVE CIARCIA

JERRY POURNELLE

EZRA SHAPIRO

BRUCE WEBSTER

SENIOR TECHNICAL EDITORS

JOHN R. EDWARDS, *Reviews*

G. MICHAEL VOSE, *Themes*

GREGG WILLIAMS, *Features*

TECHNICAL EDITORS

DENNIS ALLEN

RICHARD GREHAN

KEN SHELTON

GEORGE A. STEWART

JANE MORRILL TAZELAAR

TOM THOMPSON

CHARLES D. WESTON

EVA WHITE

STANLEY WZOLA

ASSOCIATE TECHNICAL EDITORS

CURTIS FRANKLIN, JR., *Best of BIX*

MARGARET COOK GURNEY, *Book Reviews*

BRENDA MCCLAUGHLIN, *Applications Software Reviews*
San Francisco

COPY EDITORS

BUD SADLER, *Chief*

JEFF EDMONDS

FAITH HANSON

NANCY HAYES

CATHY KINGERY

PAULA NOONAN

WARREN WILLIAMSON

JUDY WINKLER

ASSISTANTS

PEGGY DUNHAM, *Office Manager*

MARTHA HICKS

L. RYAN MCCOMBS

CHAD MITCHELL

JUNE N. SHELTON

NEWS AND TECHNOLOGY

GENE SMARTE, *Bureau Chief, Costa Mesa*

JONATHAN ERICKSON, *Senior Technical Editor, San Francisco*

RICH MALLOY, *Senior Technical Editor, New York*

CINDY KIDDOD, *Editorial Assistant, San Francisco*

ASSOCIATE NEWS EDITORS

DENNIS BARKER, *Microbytes*

CATHRYN BASKIN, *What's New*

ANNE FISCHER LENT, *What's New*

CONTRIBUTING EDITORS

JONATHAN AMSTERDAM, *programming projects*

MARK DAHMKE, *video, operating systems*

MARK HAAS, *at large*

RIK JADRNICKEK, *CAD, graphics, spreadsheets*

ROBERT T. KUROSAKA, *mathematical recreations*

PHIL LOPICCOLO, *computers in medicine*

ALASTAIR J. W. MAYER, *software*

ALAN R. MILLER, *languages and engineering*

DICK POUNTAIN, *U.K.*

ROGER POWELL, *computers and music*

WILLIAM M. RAIKE, *Japan*

PHILLIP ROBINSON, *semiconductors*

ART

NANCY RICE, *Art Director*

JOSEPH A. GALLAGHER, *Associate Art Director*

JAN MULLER, *Art Assistant*

ALAN EASTON, *Drafting*

PRODUCTION

DAVID R. ANDERSON, *Production Director*

DENISE CHARTRAND

MICHAEL J. LONSKY

VIRGINIA REARDON

TYPOGRAPHY

SHERRY MCCARTHY, *Chief Typographer*

LEN LORETTE

DONNA SWEENEY

EXECUTIVE EDITOR, BIX
GEORGE BOND

SENIOR EDITOR

DAVID BETZ

ASSOCIATE EDITORS

TONY LOCKWOOD

DONNA OSGOOD, *San Francisco*

BIX GROUP MODERATORS

DAVID P. ALLEN, *Applications Programs*

FRANK BOOSMAN, *Artificial Intelligence*

LEROY CASTERLINE, *Other*

MARC F. GREENFIELD, *Programming Languages and Tools*

JIM HOWARD, *Graphics*

GARY KENDALL, *Operating Systems*

STEVE KRENEK, *Personal Computers*

BROCK MEEKS, *Telecommunications*

BARRY NANCE, *New Technology*

DONALD OSGOOD, *Personal Computers*

SUE ROSENBERG, *Other*

JOHN SWANSON, *Chips*

BUSINESS AND MARKETING

DOUG WEBSTER, *Director, (603) 924-9027*

PATRICIA BAUSUM, *Secretary*

BRIAN WARNOCK, *Customer Service*

DENISE A. GREENE, *Customer Service*

TAMMY BURGESS, *Customer Credit and Billing*

TECHNOLOGY

CLAYTON LISLE, *Director, Business Systems Technology, MHIS*

BILL GARRISON, *Business Systems Analyst*

JACK REILLY, *Business Systems Analyst*

LINDA WOLFF, *Senior Business Systems Analyst*

McGraw-Hill Officers of McGraw-Hill Information Systems Company: President: Richard B. Miller. Executive Vice Presidents: Frederick P. Iannotti. Construction Information Group: Russell C. White. Computers and Communications Information Group: J. Thomas Ryan. Marketing and International. Senior Vice Presidents: Francis A. Shinal. Controller: Robert C. Violette. Manufacturing and Technology. Senior Vice Presidents and Publishers: Laurence Altman. Electronics Week: Harry L. Brown. BYTE; David J. McGrath. Construction Publications. Group Vice President: Peter B. McCuen. Communications Information. Vice President: Fred O. Jensen. Planning and Development.

Officers of McGraw-Hill, Inc.: Harold W. McGraw, Jr., Chairman; Joseph L. Dionne, President and Chief Executive Officer; Robert N. Landes, Executive Vice President and Secretary; Walter D. Serwatka, Executive Vice President and Chief Financial Officer; Shel F. Asen, Senior Vice President, Manufacturing; Robert J. Bahash, Senior Vice President, Finance and Manufacturing; Ralph R. Schulz, Senior Vice President, Editorial; George R. Elsing, Vice President, Circulation; Ralph J. Webb, Vice President and Treasurer.

BYTE, BITE, and The Small Systems Journal are registered trademarks of McGraw-Hill Inc.

EDITORIAL AND BUSINESS OFFICE: One Phoenix Mill Lane, Peterborough, New Hampshire 03458. (603) 924-9281.

West Coast Offices: 425 Battery St., San Francisco, CA 94111. (415) 954-9718. 3001 Red Hill Ave., Building #1, Suite 222, Costa Mesa, CA 92626. (714) 557-6292. **New York Editorial Office:** 1221 Avenue of the Americas, New York, NY 10020. (212) 512-2000.

BYTEnet: (617) 861-9764 (set modem at 8-I-N or 7-I-E; 300 or 1200 baud).

BYTE (ISSN 0360-5280) is published monthly with one extra issue per year by McGraw-Hill Inc. Founder: James H. McGraw (1860-1948). Executive, editorial, circulation, and advertising offices: One Phoenix Mill Lane, Peterborough, NH 03458, phone (603) 924-9281. Office hours: Mon-Thur 8:30 AM-4:30 PM, Friday 8:30 AM-1:00 PM, Eastern Time. Address subscriptions to BYTE Subscriptions, PO Box 590, Martinsville, NJ 08836. Postmaster: send address changes. USPS Form 3579, undeliverable copies, and fulfillment questions to BYTE Subscriptions, PO Box 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are \$21 for one year, \$38 for two years, and \$55 for three years in the U.S. and its possessions. In Canada and Mexico, \$23 for one year, \$42 for two years, \$61 for three years. \$69 for one year air delivery to Europe. \$1,000 yen for one year air delivery to Japan. \$5,600 yen for one year surface delivery to Japan. \$37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is \$3.50 in the U.S. and its possessions, \$4.25 in Canada and Mexico, \$4.50 in Europe, and \$5 elsewhere. Foreign subscriptions and sales should be remitted in U.S. funds drawn on a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

Address editorial correspondence to: Editor, BYTE, One Phoenix Mill Lane, Peterborough, NH 03458. Unacceptable manuscripts will be returned if accompanied by sufficient postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

Copyright © 1986 by McGraw-Hill Inc. All rights reserved. Trademark registered in the United States Patent and Trademark Office. Where necessary permission is granted by the copyright owner for libraries and others registered with the Copyright Clearance Center (CCC) to photocopy any article herein for the flat fee of \$1.50 per copy of the article or any part thereof. Correspondence and payment should be sent directly to the CCC, 29 Congress St., Salem, MA 01970. Specify ISSN 0360-5280/86 \$1.50. Copying done for other than personal or internal reference use without the permission of McGraw-Hill Inc. is prohibited. Requests for special permission or bulk orders should be addressed to the publisher. BYTE is available in microform from University Microfilms International, 300 North Zeeb Rd., Dept. PR, Ann Arbor, MI 48106 or 18 Bedford Row, Dept. PR, London WC1R 4EJ, England.

Subscription questions or problems should be addressed to: BYTE Subscriber Service, PO Box 328, Hancock, NH 03449.



CONTENTS
PAGE

TABLE OF CONTENTS

July	5
August	253
September	277

July

anagram.pas

Programming Project: "Anagram Solving in Pascal," by Bob Keefer. July, page 113. Also see setprob.pas.

```
program Anagram;

{Copyright 1985 by Bob Keefer}

{Anagram.Pas takes a word of up to 10 letters from
the keyboard and rearranges the letters into every
possible permutation, or anagram, of the word.}

{It then evaluates the likelihood that each anagram
is an English word by looking up every trigram in
the word in a probability table, which is stored in
a separate file PROB.DAT and is read into the array
Probability[X,Y,Z]. Finally, it records the top
scoring anagrams in Scoreboard and prints them to
the screen.}

{The program must be compiled with the Turbo
"c" compiler option to a *.COM file.}

{$A-} {compiler directive for recursion}
{$C-} {.... ignore ^C and ^S breaks}
{$I-} {.... no i/o checking}
{$V-} {.... no string checking}

const
  MaxLength = 13; {biggest word + 3}
  MaxScores = 15 ; {how many winners to store}

type
  ScoreLine = record {One line of the Scoreboard}
    Winner : string[MaxLength] ;
    Points : integer ;
  end;

var
  Word : array [1..Maxlength] of char; {Word to permute}
  Wordlength : integer; {Length of Word}
  Probability : array [0..26,0..26,0..26] of integer;
  ScoreBoard : array [1..MaxScores] of ScoreLine;
  WordToScore : string[Maxlength]; {anagram}
  DataFile : file of integer; {probability table}
  TheWord : String[Maxlength]; {Word as string}
  I : integer; {counter}

procedure Score;

var
  X,Y,Z,I,J : integer ;
  Total : integer ;
  Unlikelihood : integer;

procedure KeepScore;
```

(continued)


```

var
  N : Integer;

procedure ChalkItUp;
  var
    TempScore, I : Integer;
    TempName : String[MaxLength];

begin {ChalkItUp}
  for I := N to MaxScores do
    begin
      with ScoreBoard[I] do
        if Total > Points then
          begin
            {If an anagram}
            {scores better,}
            {then record it...}
            begin
              TempScore := Points;
              TempName := Winner;
              Points := Total;
              Winner := WordToScore
            end;
            if I <> MaxScores then
              begin {..bump the rest down}
                with ScoreBoard[I+1] do
                  begin
                    WordToScore := TempName;
                    Total := TempScore;
                  end;
                end;
              end;
            end;
          end;
        {ChalkItUp}
      end;

begin {KeepScore}
  for N := 1 to MaxScores do
    begin
      if WordToScore = ScoreBoard[N].Winner
        then Total := 0; {eliminate duplicates}
      if (Total > ScoreBoard[N].Points)
        then ChalkItUp;
      {record good-scoring words}
    end;
  end; {KeepScore}

begin {procedure Score}
  WordToScore := ' ' + WordToScore + ' ';
  Total := 0;
  Unlikelihood := 0;
  for I := 1 to length(WordToScore) - 2 do
    begin
      X := ord(copy(WordToScore,I,1))-64;
      Y := ord(copy(WordToScore,I+1,1))-64;
      Z := ord(copy(WordToScore,I+2,1))-64;
      if X < 0 then X := 0;
      if Y < 0 then Y := 0;
      if Z < 0 then Z := 0;

      Total := Total + Probability[X,Y,Z];
      if Probability[X,Y,Z] = 0 then Unlikelihood := succ(Unlikelihood);
    end;
    for J := 1 to Unlikelihood do Total := Total div 2;
  end;
  KeepScore;
end; {procedure Score}

procedure Permute (CurrentLength : Integer);

var
  I : Integer;

procedure Switch;
  var

```

```

    Temp : char;
begin
    Temp := Word[CurrentLength];
    Word[CurrentLength] := Word[I];
    Word[I] := Temp;
end; {Switch}

procedure Outword;
begin
    WordToScore := '';
    for I := 1 to Wordlength do
        WordToScore := WordToScore + Word[I];
    end; {Outword}
begin {Permute body}
    if CurrentLength = 1
    then begin
        Outword;
        Score;
    end
    else for I := 1 to CurrentLength do
        begin
            Switch;
            Permute(CurrentLength - 1);
            Switch;
        end;
    end; {Permute}

procedure GetInput;
var
    I : integer;
begin
    write('Enter word: ');
    readln(TheWord);
    WordLength := length(TheWord);
    for I := 1 to WordLength do
        begin
            Word[I] := upcase(copy(TheWord,I,1));
        end;
    TheWord := '';
    for I := 1 to WordLength do
        TheWord := TheWord + Word[I];
    end; {procedure GetInput}

procedure ZeroScore;
var I : integer;
begin
    for I:= 1 to MaxScores do
        begin
            with ScoreBoard[I] do
                begin
                    Points := 0;
                    Winner := '';
                end;
            end; {with}
        end; {ZeroScore}

procedure PostScore;
var
    I : integer;
    GotIt : boolean;
begin
    GotIt:=false;
    for I := 1 to MaxScores do

```

(continued)

```

begin
  with ScoreBoard[I] do
    begin
      if Points>0 then
        writeln(I:2, ' ', Winner, ' ', Points);
      end; {with}
    end; {for loop}
  end; {procedure PostScore}

procedure ReadProb;

var X,Y,Z : integer;

begin
  assign(Datafile,'PROB.DAT');
  reset(DataFile);
  for X := 0 to 26 do begin
    write('*');
    for Y := 0 to 26 do begin
      for Z := 0 to 26 do begin
        read(Datafile,Probability[X,Y,Z]);
      end;
    end;
  end;
  close(Datafile);
  writeln;
end; {procedure ReadProb}

procedure SignOn;
begin
  clrscr;
  writeln('Anagram.Pas');
  writeln('By Bob Keefer');
  writeln('Copyright 1985');
  writeln;
  writeln;
  writeln('To halt program, enter "*"');
  writeln;
  writeln;
  writeln;
  writeln('Reading Probability Table...');
end; {procedure Signon}

begin {Anagram program}
  SignOn;      {Display signon message}
  ReadProb;    {Read probability table}
  clrscr;
  repeat
    GetInput;           {Get word}
    ZeroScore;          {clear Scoreboard}
    Permute (Wordlength); {Evaluate words}
    writeln;
    PostScore;          {Print results}
    writeln;
    writeln;
  until Word[1]='*';
end.

```

setprob.pas

Programming Project: "Anagram Solving in Pascal," by Bob Keefer. July, page 113. Also see anagram.pas.
 Keywords: JUL86 Programming Project anagram Pascal Bob Keefer

```

program SetProb;

  {Copyright 1985 by Bob Keefer}

  {SetProb.Pas creates a 3-dimensional byte
  array, Probability[X,Y,Z], in which are stored

```


the relative probability of each 3-letter trigram found in an input text.

Once the table is completed, it is stored in the disc file PROB.DAT.}

{To use this program to add more data to an existing version of PROB.DAT, modify procedure ZeroProb so that it reads Probability[X,Y,Z] from PROB.DAT instead of zeroing the array. This can be done by commenting out the lines marked with a single * and restoring the lines marked with a double **.}

```
var
  Ch1, Ch2, Ch3 : char;
  X, Y, Z : integer;
  Filename : string[15];
  TheFile : text;
  Datafile : file of integer;
  Probability : array [0..26,0..26,0..26] of integer;

procedure ZeroProb;

var
  X,Y,Z : integer;
begin
  {assign (Datafile, 'Prob.dat');} {**}
  {reset(Datafile);} {**}
  for X:=0 to 26 do begin
    for Y:=0 to 26 do begin
      for Z:= 0 to 26 do begin
        {**} Probability[X,Y,Z] := 0;
        {**} {read(Datafile,Probability[X,Y,Z]);}
      end;
    end;
  end;
  {close (Datafile);} {**}
end; {ZeroProb}

procedure ScaleProb;

var
  X,Y,Z : integer;
begin
  for X:=0 to 26 do begin
    for Y:=0 to 26 do begin
      for Z:= 0 to 26 do begin
        Probability[X,Y,Z] :=
          (Probability[X,Y,Z] + 1)
          div 2;
      end;
    end;
  end;
end; {ScaleProb}

procedure StartUp;

begin
  clrscr;
  writeln('SetProb.Pas');
  writeln('Copyright 1985 by Bob Keefer');
  writeln;
  write ('Enter filename: ');
  readln (Filename);
  assign (TheFile, Filename);
  reset (TheFile);
end;
```

(continued)

July

```
function Cleanup ( A : integer ) : integer;
begin
  if (A>64) and (A<91) then Cleanup := A-64
    else Cleanup := 0;
end; {function Cleanup}

procedure Countem;
begin
  Ch1 := #32;
  Ch2 := #32;
  while not EOF (TheFile) do
    begin
      read(TheFile,Ch3);
      X := Cleanup(ord(upcase(Ch1)));
      Y := Cleanup(ord(upcase(Ch2)));
      Z := Cleanup(ord(upcase(Ch3)));

      if not ((X=0) and (Y=0)) or
        ((Y=0) and (Z=0))
      then Probability[X,Y,Z] :=
        Probability[X,Y,Z] + 1;
      if Probability[X,Y,Z] > 32000 then ScaleProb;
      Ch1:=Ch2;
      Ch2:=Ch3;
    end;
  end; {Countem}

procedure WriteData;
var X,Y,Z : integer;
begin
  for X := 0 to 26 do begin
    for Y := 0 to 26 do begin
      for Z := 0 to 26 do begin
        write(Datafile,Probability[X,Y,Z]);
      end;
    end;
  end;
end; {procedure WriteData}

begin {program SetProb}
  ZeroProb;
  Startup;
  Countem;
  assign(Datafile,'Prob.dat');
  rewrite(Datafile);
  WriteData;
  close(DataFile);
  close(TheFile);
  write(#7);
end.
```

chkup.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE CHKDUP - SUBROUTINE TO CHECK FOR DUPLICATED LINES
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

Mode of use:

call CHKDUP (number,flag)

where

number = integer with the element number to check.
flag = integer to show success (=0) or error (=1).

*


```

SUBTTL  FORMAL DECLARATIONS
PAGE

cschdp  SEGMENT 'CODE'
        ASSUME CS:cschdp

table   db      150 dup (0)
masks   db      1,2,4,8,16,32,64,128

SUBTTL  CHKDUP - EXECUTABLE CODE
PAGE

PUBLIC  CHKDUP
CHKDUP  PROC      FAR

        PUSH      BP
        MOV       BP,SP
        push      ds
        LDS       BX,DWORD PTR [BP+10]
        MOV       ax,[BX]
        push      ds
        mov       bx,cschdp
        mov       ds,bx
        cmp       ax,0
        je        reset
        dec       ax
        xor       dx,dx
        mov       bx,8
        div       bx
        mov       bx,offset masks
        add       bx,dx
        mov       ch,[bx]
        mov       bx,offset table
        add       bx,ax
        mov       cl,[bx]
        push      cx
        and       cl,ch
        cmp       cl,0
        jne       error
        pop       cx
        or        cl,ch
        mov       [bx],cl
        xor       ax,ax
        jmp       exit

error:   pop       cx
        mov       ax,1
        jmp       exit

reset:   mov       bx,offset table
        mov       cx,150

byteloop:
        mov       [bx],al
        inc       bx
        loop      byteloop

exit:    pop       ds
        LDS       BX,DWORD PTR [BP+6]
        mov       [bx],ax
        pop       ds
        MOV       SP,BP
        POP       BP
        RET       8

CHKDUP  ENDP
cschdp  ENDS

END

```

date.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE DATE - SUBROUTINE TO GET THE DATE
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *
DATE is a routine designed to be called from FORTRAN as a subroutine to access the date from the system.

Mode of use: call DATE (day,month,year)
where
 day = 2-byte integer containing the day of the month.
 month = 2-byte integer containing the month of the year.
 year = 2-byte integer containing the full (19xx) year value.
*

SUBTTL FORMAL DECLARATIONS
PAGE

csdate SEGMENT 'CODE'
 ASSUME CS:csdate

SUBTTL DATE - EXECUTABLE CODE
PAGE

PUBLIC date
date PROC FAR

 PUSH BP
 MOV BP,SP
 PUSH DS
; Call the system function.
 xor ax,ax
 mov ah,2Ah
 int 21h
; Handle parameters from calling program
 LDS BX,DWORD PTR [BP+6]
 MOV [BX],cx
 xor ax,ax
 mov al,dh
 LDS BX,DWORD PTR [BP+10]
 MOV [BX],ax
 mov al,dl
 LDS BX,DWORD PTR [BP+14]
 MOV [BX],ax
; Everything done except housekeeping.
exit:
 POP DS
 MOV SP,BP
 POP BP
 RET 0Ch

date ENDP
csdate ENDS

END

upcstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE UPCSTR - SUBROUTINE TO PUT A STRING IN UPPERCASE
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

UPCSTR is a routine designed to be called from FORTRAN as a subroutine to set all the alphabetic characters to uppercase.

Mode of use:

call UPCSTR (string)

where

string = name of the string (variable of type CHARACTER) to be converted to uppercase.

*

SUBTTL FORMAL DECLARATIONS

PAGE

csupcs SEGMENT 'CODE'
ASSUME CS:csupcs

SUBTTL UPCSTR - EXECUTABLE CODE

PAGE

PUBLIC UPCSTR
UPCSTR PROC FAR

PUSH BP
MOV BP,SP
PUSH ds
LDS BX,DWORD PTR [BP+6]

character:

CMP BYTE PTR [bx],0
je exit
CMP BYTE PTR [bx],6
je exit
CMP BYTE PTR [bx],'a'
JB next
CMP BYTE PTR [bx],'z'
JA next
AND BYTE PTR [bx],95

next:

INC bx
jmp character

exit:

pop ds
MOV SP,BP
POP BP
RET 4H

UPCSTR ENDP
csupcs ENDS

END

safe2sub.for

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

\$LINESIZE: 132

\$PAGESIZE: 61

\$STORAGE: 2

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C          M I C R O S A F E          C
C      Structural Analysis by Finite Elements    C
C      Module : SAFESOLV, 2nd Part              C
C      Version : 2-D                            C
C
C      COPYRIGHT (C) by MICROSTRESS Corporation - 1985,1986    C
C      ALL RIGHTS RESERVED                                C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

(continued)


```

SUBROUTINE parsfn (flspec,ddrive,fldriv,driven,flpath,flname,
+                flxtn)
C
C Parse a file specification and get drive, path, name and extension
C
  IMPLICIT INTEGER (a-z)
  CHARACTER fldriv*6,flpath*64,flname*9,flxtn*5,flspec*78,colon*2,
+          bslash*2,period*2
C
C Initialization.
C
  call setstr (78,flspec)
  call pakstr (flspec)
  call upcstr (flspec)
  fldriv=' '
  call setstr (6,fldriv)
  flpath=' '
+
  call setstr (64,flpath)
  flname=' '
  call setstr (9,flname)
  flxtn=' '
  call setstr (5,flxtn)
  colon=':'
  call setstr (2,colon)
  bslash='\'
  call setstr (2,bslash)
  period='.'
  call setstr (2,period)
C
C Determine the drive specification
C
  locatn=locstr (1,flspec,colon)
  if (locatn .eq. 0) then
    driven=ddrive+1
  else
    call movstr (fldriv,1,1,flspec,1,locatn)
    driven=ascstr (locatn-1,flspec)-64
  endif
C
C Determine the path specification
C
  firstc=locatn+1
  lastoc=locatn
10 locatn=locstr (lastoc+1,flspec,bslash)
  if (locatn .ne. 0) then
    lastoc=locatn
    goto 10
  else
    call movstr (flpath,1,1,flspec,firstc,lastoc-firstc+1)
  endif
C
C Determine the extension specification
C
  length=lenstr(flspec)
  locatn=locstr (lastoc+1,flspec,period)
  if (locatn .ne. 0) then
    call movstr (flxtn,1,1,flspec,locatn,length-locatn+1)
  else
    locatn=length+1
  endif
C
C Determine the name specification
C
  call movstr (flname,1,1,flspec,lastoc+1,locatn-lastoc-1)
C
C Pack the return strings
C
  call pakstr (fldriv)
  call pakstr (flpath)
  call pakstr (flname)
  call pakstr (flxtn)
  RETURN
END
$PAGE

```

SUBROUTINE triasemb (i,j,k,th,eyoung,pratio)

C
C
C

Assemble stiffness matrix for triangular plate

```

DOUBLE PRECISION th,diffnc(2,4),ftcons(9),eyoung,pratio
common /coordi/ coonod(2,401)
diffnc(1,2)=coonod(1,J)-coonod(1,I)
diffnc(2,2)=coonod(2,J)-coonod(2,I)
diffnc(1,3)=coonod(1,K)-coonod(1,J)
diffnc(2,3)=coonod(2,K)-coonod(2,J)
diffnc(1,1)=coonod(1,I)-coonod(1,K)
diffnc(2,1)=coonod(2,I)-coonod(2,K)
ftcons(6)=diffnc(2,3)*diffnc(1,2)-diffnc(1,3)*diffnc(2,2)
ftcons(1)=eyoung*TH/(4*ftcons(6))
ftcons(8)=ftcons(1)/(1-pratio)
ftcons(7)=ftcons(1)/(1+pratio)
ftcons(1)=ftcons(7)*
+ (diffnc(1,3)*diffnc(1,3)+diffnc(2,3)*diffnc(2,3))
ftcons(2)=ftcons(7)*
+ (diffnc(1,1)*diffnc(1,1)+diffnc(2,1)*diffnc(2,1))
ftcons(3)=ftcons(7)*
+ (diffnc(1,3)*diffnc(1,2)+diffnc(2,3)*diffnc(2,2))
ftcons(4)=ftcons(7)*ftcons(6)
ftcons(5)=ftcons(7)*
+ (diffnc(1,2)*diffnc(1,2)+diffnc(2,2)*diffnc(2,2))
I1=3*I-2
J1=3*J-2
K1=3*K-2
CALL assemble (i1,i1,ftcons(1)+ftcons(8)*diffnc(2,3)*diffnc(2,3),
+ -ftcons(8)*diffnc(1,3)*diffnc(2,3),0.)
CALL assemble (i1,j1,-ftcons(1)-ftcons(3)+
+ ftcons(8)*diffnc(2,3)*diffnc(2,1),
+ -ftcons(4)-ftcons(8)*diffnc(2,3)*diffnc(1,1),0.)
CALL assemble (i1,k1,ftcons(3)+ftcons(8)*diffnc(2,2)*diffnc(2,3),
+ ftcons(4)-ftcons(8)*diffnc(2,3)*diffnc(1,2),0.)
CALL assemble (i1+1,i1+1,ftcons(1)+
+ ftcons(8)*diffnc(1,3)*diffnc(1,3),0.,0.)
CALL assemble (i1+1,j1,ftcons(4)-ftcons(8)*
+ diffnc(2,1)*diffnc(1,3),-ftcons(1)-ftcons(3)+
+ ftcons(8)*diffnc(1,1)*diffnc(1,3),0.)
CALL assemble (i1+1,k1,
+ -ftcons(4)-ftcons(8)*diffnc(2,2)*diffnc(1,3),
+ ftcons(3)+ftcons(8)*diffnc(1,2)*diffnc(1,3),0.)
CALL assemble (j1,j1,ftcons(2)+ftcons(8)*diffnc(2,1)*diffnc(2,1),
+ -ftcons(8)*diffnc(2,1)*diffnc(1,1),0.)
CALL assemble (j1,k1,-ftcons(3)-ftcons(5)+
+ ftcons(8)*diffnc(2,1)*diffnc(2,2),
+ -ftcons(4)-ftcons(8)*diffnc(2,1)*diffnc(1,2),0.)
CALL assemble (j1+1,j1+1,
+ ftcons(2)+ftcons(8)*diffnc(1,1)*diffnc(1,1),0.,0.)
CALL assemble (j1+1,k1,ftcons(4)-
+ ftcons(8)*diffnc(1,1)*diffnc(2,2),-ftcons(3)-
+ ftcons(5)+ftcons(8)*diffnc(1,1)*diffnc(1,2),0.)
CALL assemble (K1,K1,ftcons(5)+ftcons(8)*diffnc(2,2)*diffnc(2,2),
+ -ftcons(8)*diffnc(1,2)*diffnc(2,2),0.)
CALL assemble (k1+1,k1+1,
+ ftcons(5)+ftcons(8)*diffnc(1,2)*diffnc(1,2),0.,0.)
RETURN
END

```

\$PAGE

SUBROUTINE assemble (irow,icol,add1,add2,add3)

C
C
C

Assemble the stiffness matrix

```

DOUBLE PRECISION stmtrx,stmqcn,add(3),add1,add2,add3
INTEGER longi*4
COMMON /global/ numdof,stmqcn(2,2)
common /sizebw/ malhbw
COMMON /aaaaaa/ stmtrx(8200)
add(1)=add1
add(2)=add2
add(3)=add3
do 10 i=1,3
if (add(i) .ne. 0.) then
    ic=icol+i-1
    if ((irow .le. numdof) .and. (ic .le. numdof)) then
        longi=ic+irow-1-malhbw
    end if
end if

```

(continued)


```

        if (irow .ge. ic) then
            longi=longi+malhbw*ic
        else
            longi=longi+malhbw*irow
        endif
        stmtrx(longi)=stmtrx(longi)+add(i)
    else
        longi=ic+irow-2-numdof
        if (irow .gt. numdof) then
            if (ic .le. numdof) then
                longi=longi+ic*(malhbw+1)
                stmtrx(longi)=stmtrx(longi)+add(i)
            else
                ir=irow-numdof
                icband=ic-numdof
                stmqcn(ir,icband)=stmqcn(ir,icband)+add(i)
                stmqcn(icband,ir)=stmqcn(ir,icband)
            endif
        else
            longi=longi+irow*(malhbw+1)
            stmtrx(longi)=stmtrx(longi)+add(i)
        endif
    endif
endif
ENDIF
10 continue
RETURN
END

$PAGE
SUBROUTINE triloads (inp1,inp2,inp3,th,eyoung,pratio,lpl,nodepl)
C
C Calculate forces and stresses in triangular plate
C
    DOUBLE PRECISION disdof,corfor,eyoung,pratio,th,
+      diffnc(2,4),ftcons(9)
    DIMENSION inp(3),corfor(2,3),nodepl(4,500)
    INTEGER previd
    common /coordi/ coonod(2,401)
    COMMON /plates/ disdof(1203),pltecf(2,4),plstrs(3,500),
+      reafor(3,400),pstnor(3,400),pstacc(3,400)
    previd(k,l)=MOD(k+l-2,l)+1
    nextid(k,l)=MOD(k,l)+1
    inp(1)=inp1
    inp(2)=inp2
    inp(3)=inp3
    I=nodepl(inp(1),LPL)
    J=nodepl(inp(2),LPL)
    IF (inp(3) .lt. 0) THEN
        K=-inp(3)
        nan=2
    ELSE
        K=nodepl(inp(3),LPL)
        nan=3
    ENDIF
    I1=3*I-2
    J1=3*J-2
    K1=3*K-2
    diffnc(1,2)=coonod(1,J)-coonod(1,I)
    diffnc(2,2)=coonod(2,J)-coonod(2,I)
    diffnc(1,3)=coonod(1,K)-coonod(1,J)
    diffnc(2,3)=coonod(2,K)-coonod(2,J)
    diffnc(1,1)=coonod(1,I)-coonod(1,K)
    diffnc(2,1)=coonod(2,I)-coonod(2,K)
    ftcons(4)=eyoung/((1+pratio)*(diffnc(1,1)*diffnc(2,2)-
+      diffnc(1,2)*diffnc(2,1)))
    ftcons(5)=diffnc(2,3)*disdof(I1)+diffnc(2,1)*disdof(J1)+
+      diffnc(2,2)*disdof(K1)
    ftcons(6)=diffnc(1,3)*disdof(I1+1)+diffnc(1,1)*disdof(J1+1)+
+      diffnc(1,2)*disdof(K1+1)
    ftcons(1)=(pratio*ftcons(6)-ftcons(5))*ftcons(4)/(1-pratio)
    ftcons(2)=(ftcons(6)-pratio*ftcons(5))*ftcons(4)/(1-pratio)
    ftcons(3)=(diffnc(1,3)*disdof(I1)-diffnc(2,3)*disdof(I1+1)+
+      diffnc(1,1)*disdof(J1)-diffnc(2,1)*disdof(J1+1)+
+      diffnc(1,2)*disdof(K1)-diffnc(2,2)*disdof(K1+1))*
+      ftcons(4)/2
    DO 20 LL=1,NAN
        INDX=nodepl(inp(LL),LPL)

```

```

      ftcons(7)=ABS(diffnc(2,nextid(LL,3))-diffnc(2,LL))/
+      (ABS(diffnc(1,nextid(LL,3))-diffnc(1,LL))+
+      ABS(diffnc(2,nextid(LL,3))-diffnc(2,LL)))
      DO 10 L=1,2
      corfor(L,LL)=TH*.5*(diffnc(1,previ(LL,3))*ftcons(4-L)-
+      diffnc(2,previ(LL,3))*ftcons(2*L-1))
      pltecf(L,inp(LL))=pltecf(L,inp(LL))+corfor(L,LL)
      ftcons(7)=1-ftcons(7)
      reafor(L,INDX)=reafor(L,INDX)+corfor(L,LL)
      pstnor(L,INDX)=pstnor(L,INDX)+ftcons(7)
      pstacc(L,INDX)=pstacc(L,INDX)+ftcons(7)*ftcons(L)
10  CONTINUE
      pstacc(3,INDX)=pstacc(3,INDX)+ftcons(3)
      pstnor(3,INDX)=pstnor(3,INDX)+1
      plstrs(LL,LPL)=plstrs(LL,LPL)+ftcons(LL)
20  CONTINUE
      IF (nan .EQ. 2) plstrs(3,LPL)=plstrs(3,LPL)+ftcons(3)
      RETURN
      END
$PAGE
      SUBROUTINE opnfil (ierror)
C
C  Open a file for output with verification
C
      LOGICAL ffound
      CHARACTER inpfil*78,outfil*78,prompt*55,intgst*25
      common /filnm/ inpfil,outfil
      inquire (FILE=outfil,EXIST=ffound)
      if (.not.(ffound)) then
        call setstr (78,outfil)
        call pakstr (outfil)
        length=lenstr(outfil)+1
        call expstr (outfil)
        call resstr (outfil)
        call setstr (length,outfil)
        call chopwr (outfil,ierror)
        if (ierror .ne. 0) then
          call resstr (outfil)
          length=length-1
          call wrfstr (float(length),intgst)
          length=lenstr (intgst)
          prompt='(' ERROR : File "'',a ,'" cannot be open. Try a
+gain.')
          call setstr (55,prompt)
          call movstr (prompt,21,0,intgst,1,length)
          call resstr (prompt)
          write (*,prompt) outfil
          return
        endif
        call resstr (outfil)
      endif
      OPEN (2,FILE=outfil,STATUS='new')
      ierror=0
      return
      END
$PAGE
      SUBROUTINE diskroom (nbytes)
C
C  Update count of characters in output file to avoid disk full errors.
C
      INTEGER frespc*4,odrive,scrflg,ascilc
      COMMON /diskrom/ scrflg,odrive
C
      if (nbytes .eq. 0) then
        call dskepc (odrive,frespc)
        frespc=frespc-1
      else
C
C  20   frespc=frespc-nbytes
C
        if (frespc .lt. 0) then
          close (2)
          ascilc=odrive+64
          write (*,30)
30      format (// ' ERROR : Output file disk is full.')

```

(continued)

[illegible]

```

+          , 'coordinates of node      '
+          , 'properties of material code '
+          , 'properties of beam        '
+          , 'properties of plate       '
+          , 'properties of fastener    '
+          , 'applied loads to node     '
+          , 'imposed displacements to node '
DATA ordind / 'first '
+          , 'second '
+          , 'third '
+          , 'fourth '
+          , 'fifth '
+          , 'sixth '
+          , 'seventh '
+          , 'eighth ' /
DATA itxtpr / 1,2,3,4,5,6,7,8,11,14,22,29,34,38,
+          0,0,0,0,0,0,0,9,12,15,23,30,35,39,
+          0,0,0,0,0,0,0,10,13,16,24,31,36,40,
+          0,0,0,0,0,0,0,0,17,25,32,37,0,
+          0,0,0,0,0,0,0,0,18,26,33,0,0,
+          0,0,0,0,0,0,0,0,19,27,0,0,0,
+          0,0,0,0,0,0,0,0,20,28,0,0,0,
+          0,0,0,0,0,0,0,0,21,0,0,0,0/
stcons='
line='
+
slash='/'
call setstr(2,slash)
space=' '
call setstr(2,space)
grafch=char(9)
tabchr=' '
call setstr(2,tabchr)
call movstr(tabchr,1,0,grafch,1,1)
chrerr=0
idparm=1
locatn=1
if (idline .eq. 1) linumb=0
10 linumb=linumb+1
ierror=1
READ (1,20,END=70,ERR=1000) buffer
20 FORMAT (A126)
call setstr(126,buffer)
ierror=0
ENDSEP=locstr(1,buffer,slash)
IF (ENDSEP .eq. 0) goto 10
call endstr (endsep+1,buffer)
25 itcons=locstr(locatn,buffer,tabchr)
if (itcons .ne. 0) then
    call movstr (buffer,itcons,0,space,1,1)
    locatn=itcons+1
    goto 25
endif
locatn=1
30 IF (locatn .ge. ENDSEP) THEN
    chrerr=ENDSEP
    ierror=2
    GOTO 70
endif
seprtr=locstr(locatn,buffer,space)
IF (seprtr .eq. locatn) THEN
    locatn=locatn+1
    GOTO 30
endif
IF ((seprtr .eq. 0) .OR. (seprtr .gt. ENDSEP)) seprtr=ENDSEP
ierror=0
decpop=0
EXPFLG=0
EXPSGN=0
seploc=seprtr-locatn
do 50 positn=1,SEPLOC
    index=locatn+positn-1
    asciic=ascstr(index,buffer)
    IF ((asciic .gt. 47) .AND. (asciic .lt. 58)) goto 40
    IF ((positn .eq. 1) .AND. ((asciic .eq. 43) .OR.

```

(continued)


```

+ (asciic .eq. 45))) goto 40
IF (((asciic .eq. 46) .AND. (decpop .eq. 0)) THEN
    decpop=locatn+positn-1
    GOTO 40
endif
IF (((asciic .eq. 68) .OR. (asciic .eq. 69) .OR. (asciic .eq. 100)
+ .OR. (asciic .eq. 101)) .AND. (EXPFLG .eq. 0)) THEN
    EXPFLG=locatn+positn
    GOTO 40
endif
IF (((asciic .eq. 43) .OR. (asciic .eq. 45)) .AND. (EXPFLG .ne. 0)
+ .AND. (EXPSGN .eq. 0)) THEN
    EXPSGN=locatn+positn
    if (asciic .gt. 43) expsgn=-expsgn
    GOTO 40
endif
ierror=3
chrerr=locatn+positn-1
goto 60
40 continue
50 continue
60 continue
IF (ierror .eq. 3) goto 70
call setstr(25,stcons)
call movstr(stcons,1,1,buffer,locatn,SEPLOC)
call resstr(stcons)
ftcons=fltstr(stcons)
IF ((ftcons .lt. boulow(idline,idparm)) .OR.
+ (ftcons .gt. bouhig(idline,idparm))) THEN
    ierror=5
    chrerr=locatn
    GOTO 70
endif
IF ((itypar(idline,idparm) .eq. 1) .and.
+ (ftcons .ne. float(int(ftcons)))) then
    ierror=4
    IF (decpop .ne. 0) THEN
        chrerr=decpop
        GOTO 70
    ELSE
        IF (EXPSGN .lt. 0) THEN
            chrerr=-EXPSGN
            GOTO 70
        ELSE
            chrerr=locatn
            GOTO 70
        endif
    endif
endif
entry(idparm)=ftcons
if ((idparm .eq. 1) .and. (idline .gt. 7) .and. (idline .lt. 14))
+ then
    itcons=INT(ftcons)
    CALL CHKDUP (itcons,ierror)
    IF (ierror .ne. 0) THEN
        ierror=9
        chrerr=locatn
        goto 70
    endif
else
    if ((idparm .eq. 2) .and. (idline .eq. 14)) then
        itcons=INT(3*entry(1)+ftcons-3)
        CALL CHKDUP (itcons,ierror)
        IF (ierror .ne. 0) THEN
            ierror=9
            chrerr=locatn
            goto 70
        endif
    endif
endif
locatn=seprtr+1
idparm=idparm+1
IF (idparm .gt. numpar(idline)) THEN
    if (idline .lt. 6) bouhig(idline+7,1)=entry(1)
    if (idline .eq. 1) then
        bouhig(10,2)=entry(1)
    endif
endif

```



```

        bouhig(10,3)=entry(1)
        bouhig(11,2)=entry(1)
        bouhig(11,3)=entry(1)
        bouhig(11,4)=entry(1)
        bouhig(11,5)=entry(1)
        bouhig(12,2)=entry(1)
        bouhig(12,3)=entry(1)
        bouhig(13,1)=entry(1)
        bouhig(14,1)=entry(1)
    endif
    if (idline .eq. 2) then
        bouhig(10,6)=entry(1)
        bouhig(11,7)=entry(1)
    endif
    if (((idline .eq. 10) .and. (entry(4) .ne. 0.) .and.
+      (youngm(int(entry(6))) .ne. 0.)) .or.
+      ((idline .eq. 12) .and. (entry(5) .ne. 0.))) then
        nod1=int(entry(2))
        nod2=int(entry(3))
        lbanwd=3*(1+abs(nod1-nod2))
        if (lbanwd .gt. malhbw) then
            ierror=6
            goto 70
        else
            if (lbanwd .eq. 3) then
                ierror=7
                goto 70
            endif
        endif
        if (idline .eq. 10) then
            if ((coonod(1,nod1) .eq. coonod(1,nod2)) .and.
+          (coonod(2,nod1) .eq. coonod(2,nod2))) then
                ierror=8
                goto 70
            endif
        endif
        if (lbanwd .gt. maxban) maxban=lbanwd
    else
        if (((idline .eq. 11) .and. (entry(6) .ne. 0.) .and.
+      (youngm(int(entry(7))) .ne. 0.)) then
            maxnod=max(int(entry(2)),int(entry(3)),
+          int(entry(4)))
            minnod=min(int(entry(2)),int(entry(3)),
+          int(entry(4)))
            if (entry(5) .ne. 0.) then
                maxnod=max(maxnod,int(entry(5)))
                minnod=min(minnod,int(entry(5)))
            endif
            lbanwd=3*(1+maxnod-minnod)
            if (lbanwd .gt. malhbw) then
                ierror=6
                goto 70
            endif
            do 65 itcons=2,4
                nod1=int(entry(itcons))
                startp=itcons+1
                do 65 index=startp,5
                    nod2=int(entry(index))
                    if (nod2 .ne. 0) then
                        if (nod1 .eq. nod2) then
                            ierror=7
                            goto 70
                        else
+                            if ((coonod(1,nod1) .eq. coonod(1,nod2)) .and.
                                (coonod(2,nod1) .eq. coonod(2,nod2))) then
                                    ierror=8
                                    goto 70
                                endif
                            endif
                        endif
                    endif
                continue
            if (lbanwd .gt. maxban) maxban=lbanwd
        endif
    endif
    goto 3000

```

(continued)

```

ELSE
    goto 30
endif
70 txtpar(1)='number of nodes in the model
txtpar(2)='number of types of materials in the model
txtpar(3)='number of beams in the model
txtpar(4)='number of plates in the model
txtpar(5)='number of fasteners in the model
txtpar(6)='number of loaded nodes in the model
txtpar(7)='number of restrained displacements in the model
txtpar(8)='node number
txtpar(9)='x coordinate of the node
txtpar(10)='y coordinate of the node
txtpar(11)='material number
txtpar(12)='Young's modulus of the material
txtpar(13)='Poisson's ratio of the material
txtpar(14)='beam number
txtpar(15)='index of the first node of the beam
txtpar(16)='index of the second node of the beam
txtpar(17)='beam area
txtpar(18)='beam moment of inertia
txtpar(19)='beam material code
txtpar(20)='distributed load at the first node of the beam
txtpar(21)='distributed load at the second node of the beam
txtpar(22)='plate number
txtpar(23)='index of the first node of the plate
txtpar(24)='index of the second node of the plate
txtpar(25)='index of the third node of the plate
txtpar(26)='index of the fourth node of the plate
txtpar(27)='plate thickness
txtpar(28)='plate material code
txtpar(29)='fastener number
txtpar(30)='index of the first node of the fastener
txtpar(31)='index of the second node of the fastener
txtpar(32)='fastener area
txtpar(33)='fastener stiffness
txtpar(34)='loaded node number
txtpar(35)='applied load at the node along the x direction
txtpar(36)='applied load at the node along the y direction
txtpar(37)='applied moment at the node along the z direction
txtpar(38)='node number with a restrained degree of freedom
txtpar(39)='restrained degree of freedom of the node
txtpar(40)='imposed displacement at the node
write (*,80) errmsg(ierr)
80 FORMAT (// ' ERROR : ',A50)
call diskroom (67)
write (2,80,err=2000) errmsg(ierr)
messge(1)=
+
+ messge(2)=
+
+ messge(3)=
+
call setstr (240,MESSGE(1))
stcons='Encountered
call movstr (messge(1),1,1,stcons,1,11)
IF (ierror .eq. 1) THEN
    stcons=' attempting to read
ELSE
    stcons=' in
endif
call setstr (25,stcons)
call constr (messge(1),stcons)
call pakstr (messge(1))
stcons=' line
call setstr (6,stcons)
call constr (messge(1),stcons)
call pakstr (messge(1))
call constr (messge(1),space)
call wrfstr (float(linum),stcons)
call constr (messge(1),stcons)
call pakstr (messge(1))
stcons=' of file
call setstr (9,stcons)
call constr (messge(1),stcons)
call constr (messge(1),space)

```



```

call setstr (78,inpfil)
call pakstr (inpfil)
call constr (messge(1),inpfil)
period='.'
call setstr (2,period)
call constr (messge(1),period)
call writxt (messge)
IF (ierror .eq. 1) goto 3000
grafch=char(218)
call setstr (79,line)
call filstr (196,line)
call movstr (line,1,0,grafch,1,1)
if (chrerr .ne. 0) then
    grafch=char(25)
    call movstr (line,chrerr+1,0,grafch,1,1)
endif
length=lenstr (buffer)+2
grafch=char(191)
call movstr (line,length,0,grafch,1,1)
length=length+1
call endstr (length,line)
call resstr (line)
write (*,90) line
90 format (1x,A79)
call diskroom (82)
write (2,90,err=2000) line
length=length-3
call setstr (79,line)
grafch=char(179)
call movstr (line,1,0,grafch,1,1)
call movstr (line,2,0,buffer,1,length)
length=length+2
call movstr (line,length,0,grafch,1,1)
length=length+1
call endstr (length,line)
call resstr (line)
write (*,90) line
call diskroom (82)
write (2,90,err=2000) line
grafch=char(192)
call setstr (79,line)
call filstr (196,line)
call movstr (line,1,0,grafch,1,1)
if (chrerr .ne. 0) then
    grafch=char(24)
    call movstr (line,chrerr+1,0,grafch,1,1)
endif
length=lenstr (buffer)+2
grafch=char(217)
call movstr (line,length,0,grafch,1,1)
length=length+1
call endstr (length,line)
call resstr (line)
write (*,90) line
call diskroom (82)
write (2,90,err=2000) line
call filstr (32,messge(1))
if (ierror .eq. 6) then
    stcons=' The bandwidth for '
    call movstr (messge(1),1,0,stcons,1,18)
    call movstr (messge(1),20,0,lintyp(idline),1,16)
    call pakstr (messge(1))
    call constr (messge(1),space)
    call wrfstr (entry(1),stcons)
    call constr (messge(1),stcons)
    stcons=' is '
    call setstr (4,stcons)
    call constr (messge(1),stcons)
    call constr (messge(1),space)
    call wrfstr (float(lbanwd),stcons)
    call constr (messge(1),stcons)
    stcons=' and exceeds the maximum '
    call setstr (25,stcons)
    call constr (messge(1),stcons)
    stcons=' allowed bandwidth of '

```

(continued)

```

call setstr (22,stcons)
call constr (messge(1),stcons)
call constr (messge(1),space)
call wrfstr (float(malhbw),stcons)
call constr (messge(1),stcons)
call constr (messge(1),period)
call writxt (messge)
goto 3000
endif
if (ierror .eq. 7) then
stcons=' There are identical node'
call movstr (messge(1),1,0,stcons,1,25)
call pakstr (messge(1))
stcons='s in '
call setstr (5,stcons)
call constr (messge(1),stcons)
call constr (messge(1),space)
call setstr (16,lintyp(idline))
call constr (messge(1),lintyp(idline))
call resstr (lintyp(idline))
call pakstr (messge(1))
call constr (messge(1),space)
call wrfstr (entry(1),stcons)
call constr (messge(1),stcons)
call constr (messge(1),period)
call writxt (messge)
goto 3000
endif
if (ierror .eq. 8) then
stcons=' Nodes '
call movstr (messge(1),1,0,stcons,1,6)
call pakstr (messge(1))
call constr (messge(1),space)
call wrfstr (float(nod1),stcons)
call constr (messge(1),stcons)
stcons=' and '
call setstr (5,stcons)
call constr (messge(1),stcons)
call constr (messge(1),space)
call wrfstr (float(nod2),stcons)
call constr (messge(1),stcons)
stcons=' of '
call setstr (5,stcons)
call constr (messge(1),stcons)
call setstr (16,lintyp(idline))
call constr (messge(1),lintyp(idline))
call resstr (lintyp(idline))
call pakstr (messge(1))
call constr (messge(1),space)
call wrfstr (entry(1),stcons)
call constr (messge(1),stcons)
stcons=' have the same coordinat '
call setstr (25,stcons)
call constr (messge(1),stcons)
stcons='es. '
call setstr (4,stcons)
call constr (messge(1),stcons)
call writxt (messge)
goto 3000
endif
if (ierror .eq. 9) then
stcons=' The '
call movstr (messge(1),1,0,stcons,1,5)
call pakstr (messge(1))
call constr (messge(1),space)
call setstr (30,linent(idline))
call constr (messge(1),linent(idline))
call resstr (linent(idline))
call pakstr (messge(1))
call constr (messge(1),space)
call wrfstr (entry(1),stcons)
call constr (messge(1),stcons)
stcons=' appear twice. '
call setstr (15,stcons)
call constr (messge(1),stcons)

```



```

        call writxt (messge)
        goto 3000
    endif
    stcons=' Reading
    call movstr (messge(1),1,0,stcons,1,8)
    if (idparm .eq. 1) then
        call movstr (messge(1),10,0,lintyp(idline),1,16)
        call pakstr (messge(1))
        stcons=' lines
        call setstr (7,stcons)
        call constr (messge(1),stcons)
    else
        call movstr (messge(1),10,0,linent(idline),1,30)
        call pakstr (messge(1))
        call constr (messge(1),space)
        call wrfstr (entry(1),stcons)
        call constr (messge(1),stcons)
    endif
    stcons=' it was expected to find '
    call setstr(25,stcons)
    call constr(messge(1),stcons)
    if ((idparm .eq. 1) .and. (idline .gt. 7)) then
        stcons=' a
    else
        stcons=' the
    endif
    call setstr (5,stcons)
    call constr (messge(1),stcons)
    call pakstr (messge(1))
    call constr (messge(1),space)
    index=itxtpr(idline,idparm)
    call setstr (49,txtpar(index))
    call constr (messge(1),txtpar(index))
    call resstr (txtpar(index))
    call pakstr (messge(1))
    index=itypar(idline,idparm)
    call setstr (14,typpar(index))
    call constr (messge(1),typpar(index))
    call resstr (typpar(index))
    call pakstr (messge(1))
    stcons=' between
    call setstr (10,stcons)
    call constr (messge(1),stcons)
    call wrfstr (boulow(idline,idparm),stcons)
    call constr (messge(1),stcons)
    stcons=' and
    call setstr (6,stcons)
    call constr (messge(1),stcons)
    call wrfstr (bouhig(idline,idparm),stcons)
    call constr (messge(1),stcons)
    stcons=' - as the
    call setstr (11,stcons)
    call constr (messge(1),stcons)
    call setstr (8,ordinl(idparm))
    call constr (messge(1),ordinl(idparm))
    call resstr (ordinl(idparm))
    call pakstr (messge(1))
    stcons=' entry.
    call setstr (8,stcons)
    call constr (messge(1),stcons)
    call writxt (messge)
    goto 3000
1000 write (*,1010)
1010 format (//' ERROR : CANNOT READ INPUT FILE.'/
+         ' The program cannot continue.')
    ierror=-1
    goto 3000
2000 write (*,2010)
2010 format (//' ERROR : CANNOT WRITE OUTPUT FILE.'/
+         ' The program cannot continue.')
    ierror=-1
3000 return
end
$PAGE

```

(continued)

```

SUBROUTINE writxt (messge)
C
C Write text on the screen formatting to avoid breaking words
C
  IMPLICIT INTEGER (a-z)
  CHARACTER messge*80,line*79,endwrd*3,space*2
  DIMENSION messge(3)
  line='
+
  call setstr (79,line)
  endwrd='
  call setstr (3,endwrd)
  space='
  call setstr (2,space)
  call expstr (messge(1))
  startp=1
  endtxt=locstr (1,messge(1),endwrd)
110 index=startp+79
  IF (ENDTXT .ge. index) THEN
    spcpos=startp-1
120    ntxtspc=spcpos+1
    length=locstr (ntxtspc,messge(1),space)
    IF (length .lt. index) THEN
      spcpos=length
      GOTO 120
    endif
    length=spcpos-startp
    call movstr (line,1,1,messge(1),startp,length)
    call resstr (line)
    write (*,90) line
90    format (1x,A79)
    call diskroom (82)
    write (2,90,err=2000) line
    call setstr (79,line)
    startp=spcpos+1
    GOTO 110
  endif
  endtxt=endtxt-1
  call movstr (line,1,1,messge(1),startp,ENDTXT)
  call resstr (line)
  write (*,90) line
  call diskroom (82)
  write (2,90,err=2000) line
  goto 3000
2000 write (*,2010)
2010 format (//' ERROR : CANNOT WRITE OUTPUT FILE.'/
+          ' The program cannot continue.')
  ierror=-1
3000 return
end

$PAGE
FUNCTION degree (oppsid,closid)
C
C Determine angle in degrees with opposite and next side of triangle.
C
  IF (abs(closid) .gt. 1e-19) THEN
    degree=57.2957795*ATAN(oppsid/closid)
    IF (closid .LT. 0.) degree=degree+180.
    IF (degree .gt. 180.) degree=degree-360.
  ELSE
    IF (oppsid .ge. 0.) then
      degree=90.
    else
      degree=-90.
    endif
  ENDIF
  RETURN
END

$PAGE
SUBROUTINE datstr(string)
C
C Write the date in a string.
C
  IMPLICIT integer (a-z)
  CHARACTER string*11,blank*2,buffer*10
  call date (day,month,year)
  write (buffer,10) month,day,year

```



```

10 FORMAT (i2,'/',i2,'/',i4)
   READ (buffer,20) string
20 format (a10)
   call setstr (11,string)
   asciic=ascstr(4,string)
   if (asciic .eq. 32) call modstr (string,4,48)
   RETURN
   END
$PAGE
   SUBROUTINE timstr(string)
C
C Write the time-of-day in a string.
C
   IMPLICIT integer (a-z)
   real realsc
   CHARACTER string*12,blank*2,buffer*11
   call time (hour,minute,second,sec100)
   realsc=float(second)+float(sec100)/100.
   write (buffer,10) hour,minute,realsc
10 FORMAT (i2,':',i2,':',f5.2)
   READ (buffer,20) string
20 format (a11)
   call setstr (12,string)
   asciic=ascstr(4,string)
   if (asciic .eq. 32) call modstr (string,4,48)
   asciic=ascstr(7,string)
   if (asciic .eq. 32) then
       call modstr (string,7,48)
       asciic=ascstr(8,string)
       if (asciic .eq. 32) call modstr (string,8,48)
   endif
   RETURN
   END
$PAGE
   FUNCTION fltstr (string)
C
C Calculate the floating point value of a string.
C
   CHARACTER buffer*26,string*25
   write (buffer,*) string
   READ (buffer,10,ERR=300) intstr
10 format (bn,i25)
   fltstr=float(intstr)
   goto 500
300 fltstr=0
   READ (buffer,310,ERR=500) fltstr
310 format (bn,f25.0)
500 RETURN
   END
$PAGE
   SUBROUTINE wrfstr (real,string)
C
C Write a real in a string.
C
   implicit integer (a-z)
   real real
   CHARACTER string*25,expnnt*5
   if (real .eq. 0.) then
       string='0'
       call setstr (25,string)
       call endstr (2,string)
   else
       if ((abs(real) .ge. 1.e11) .or. (abs(real) .lt. 1.e-5)) then
           write (string,10) real
10          format (E12.6E2)
           call setstr (25,string)
           call pakstr (string)
           expnnt='E'
           call setstr (5,expnnt)
           call endstr (2,expnnt)
           l=locstr (1,string,expnnt)
           call movstr (expnnt,1,1,string,l,4)
30          l=l-1
           if (ascstr(l,string) .eq. 48) goto 30
           call movstr (string,l+1,1,expnnt,l,4)

```

(continued)

```

else
40      write (string,40) real
        format (F19.10)
        call setstr (25,string)
        call pakstr (string)
        l=lenstr (string)+1
50      l=l-1
        if (ascstr(l,string) .eq. 48) goto 50
        if (ascstr(l,string) .eq. 46) l=l-1
        call endstr (l+1,string)
endif
endif
RETURN
END

```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

28 BYTE LISTINGS SUPPLEMENT • JULY-SEPTEMBER, 1986


```
COMMON /aaaaaa/ stmx(8200)
common /filenm/ inpfil,outfil
common /forces/ appldf(1200)
COMMON /dskrom/ scrflg,odrive
common /coordi/ coonod(2,401)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     USER DEFINED FUNCTIONS                                C
C                                                                                             C
C                                     GENERAL INITIALIZATION                               C
C                                                                                             C
call time (inithr,initmn,initsc,iniths)
call datstr (datext)
call timstr (timtxt)
C
C Show copyright notice on the screen.
C
call logpsl
C
Initialize variables.
C
scrflg=0
maxban=6
stmqcn(1,1)=0.
stmqcn(1,2)=0.
stmqcn(2,1)=0.
stmqcn(2,2)=0.
space=' '
call setstr (2,space)
toextn='.OUT'
call setstr (5,toextn)
elipss='...'
call setstr (4,elipss)
call defdrv (0,ddrive)
C
Determine number of stiffness matrix elements which will fit in RAM.
C
numele=memava(stmx(1))/4
if (numele .gt. 65535) numele=65535
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     READ THE COMMAND TAIL                                C
C                                                                                             C
C                                     +
error=ppmuqq (0,0,comand)
length=ascstr (1,comand)+2
if (length .ne. 2) then
    call setstr (127,comand)
    call endstr (length,comand)
    call movstr (comand,1,0,space,1,1)
    call upcstr (comand)
    string=' I = '
    call setstr (4,string)
    locatn=locstr (1,comand,string)+3
    if (locatn .ne. 3) then
        nxtloc=locstr (locatn,comand,space)
        if (nxtloc .eq. 0) nxtloc=length
        numchr=nxtloc-locatn
        inpfil='
            call setstr (78,inpfil)
            call movstr (inpfil,1,0,comand,locatn,numchr)
            call resstr (inpfil)
            ifnflg=1
        endif
        call modstr (string,2,79)
        locatn=locstr (1,comand,string)+3
        if (locatn .ne. 3) then
            nxtloc=locstr (locatn,comand,space)
```

(continued)

[illegible]


```

      call resstr (prompt)
      WRITE (*,prompt) toufil
      READ (*,'(A)') outfil
    else
      call resstr (prompt)
      WRITE (*,prompt) toufil,outfil
    endif
    flspec=outfil
    call parsfn (flspec,idrive-1,ofdriv,odrive,opath,ofname,ofextn)
    outfil=flspec
    IF (lenstr(ofdriv).le. 2) then
      call setstr (78,outfil)
      call endstr (1,outfil)
      if (lenstr(ofdriv).eq. 0) ofdriv=idriv
      if (lenstr(opath).eq. 0) opath=ipath
      if (lenstr(ofname).eq. 0) ofname=ifname
      if (lenstr(ofextn).eq. 0) ofextn=toextn
      call constr (outfil,ofdriv)
      call constr (outfil,opath)
      call constr (outfil,ofname)
      call constr (outfil,ofextn)
    endif
    call resstr (outfil)
    call opnfil (ierror)
    if (ierror.ne. 0) then
      ofnflg=0
      goto 74
    endif
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     START THE OUTPUT FILE                                C
C                                                                                          C
C                                     call diskroom (0)                                  C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     Header title                                         C
C                                                                                          C
C                                     call diskroom (331)                                 C
C                                     WRITE (2,80,err=2000) datetxt,timtxt,inpfil,outfil   C
80 FORMAT (' M I C R O S A F E --- STRUCTURAL ANALYSIS BY FINITE EL ',          C
+ 'EMENTS',4x,'Version: SAFESOLV (2-D)',2x,'Rel. 1.0',3x,a10,1x,a8//           C
+/ ' Input data file : ',A/' Output data file : ',A/)                            C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     START READING THE INPUT FILE                        C
C                                                                                          C
C                                     diamsg='Reading model data from file              C
+                                                                                                                                 C
      call setstr (110,diamsg)
      call setstr (78,inpfil)
      call movstr (diamsg,30,0,inpfil,1,77)
      call resstr (inpfil)
      call pakstr (diamsg)
      call constr (diamsg,elipss)
      call expstr (diamsg)
      call resstr (diamsg)
      call resstr (ofdriv)
      if (ofdriv.eq. 'CON:') scrflg=-1
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C                                     READ AND PROCESS THE MODEL SIZE LINES                C
C                                                                                          C
C                                     Size header                                          C
C                                                                                          C
      if (echflg.eq. 1) then
        if (scrflg.eq. 1) then
95         WRITE (*,85)
            FORMAT (/ ' SIZE OF THE STRUCTURE' / )
        else
          if (scrflg.eq. 0) write (*,87) diamsg
97         format (/ 1X,A/ ' Size... \ ')
        endif

```

(continued)

```

        call diskroom (30)
        WRITE (2,85,err=2000)
    else
        write (*,87) diamsg
    endif
C
C Number of nodes and degrees of freedom
C
    CALL verify(1,entry,ierror,maxban,youngm)
    IF (ierror .NE. 0) GOTO 994
    nnodes=entry(1)
    if (echflg .eq. 1) then
        if (scrflg .eq. 1) WRITE (*,90) nnodes
90    FORMAT (' Number of nodes',I4)
        call diskroom (48)
        WRITE (2,90,err=2000) nnodes
    endif
    DO 92 loop=1,nnodes
    DO 92 indx=1,3
        reafor(indx,loop)=0.
        pstnor(indx,loop)=0.
92    pstacc(indx,loop)=0.
        numdof=3*nnodes
    DO 94 loop=1,numdof
94    appldf(loop)=0.
        malhbw=numele/numdof-2
        if (malhbw .gt. numdof) malhbw=numdof
        longj=numdof*(malhbw+2)
        do 96 longi=1,longj
96    stmtrx(longi)=0.
C
C Number of types of material
C
    CALL verify(2,entry,ierror,maxban,youngm)
    IF (ierror .NE. 0) GOTO 994
    nmater=entry(1)
    if (echflg .eq. 1) then
        if (scrflg .eq. 1) WRITE (*,98) nmater
98    FORMAT (' Number of materials',I4)
        call diskroom (48)
        WRITE (2,98,err=2000) nmater
    endif
C
C Number of beams
C
    CALL verify(3,entry,ierror,maxban,youngm)
    IF (ierror .NE. 0) GOTO 994
    nbeams=entry(1)
    if (echflg .eq. 1) then
        if (scrflg .eq. 1) WRITE (*,100) nbeams
100    FORMAT (' Number of beams',I4)
        call diskroom (48)
        WRITE (2,100,err=2000) nbeams
    endif
C
C Number of plates
C
    CALL verify(4,entry,ierror,maxban,youngm)
    IF (ierror .NE. 0) GOTO 994
    nplate=entry(1)
    if (echflg .eq. 1) then
        if (scrflg .eq. 1) WRITE (*,105) nplate
105    FORMAT (' Number of plates',I4)
        call diskroom (48)
        WRITE (2,105,err=2000) nplate
    endif
    DO 107 loop=1,nplate
    DO 107 indx=1,3
107    plstrs(indx,loop)=0.
C
C Number of fasteners
C
    CALL verify(5,entry,ierror,maxban,youngm)
    IF (ierror .NE. 0) GOTO 994
    nfastn=entry(1)
    if (echflg .eq. 1) then

```


BYTE LISTINGS SUPPLEMENT • JULY-SEPTEMBER, 1986 33

[illegible]

[illegible]

(continued)

```

240      write (*,240)
      FORMAT (//' PLATE DATA'//' Plate   I       J       K',
+          '      L   Thickness Material'//)
      else
245      if (scrflg .eq. 0) write (*,245)
          format ('Plates...'\\)
      endif
      call diskroom (78)
      WRITE (2,240,err=2000)
      else
          write (*,245)
      endif
      call chkdup (0,ierror)
      DO 360,loop=1,nplate
      CALL verify(11,entry,ierror,maxban,youngm)
      IF (ierror .NE. 0) GOTO 994
      i=entry(1)
      n1=entry(2)
      n2=entry(3)
      n3=entry(4)
      n4=entry(5)
      pthick=entry(6)
      mat=entry(7)
      if (echflg .eq. 1) then
250      if (scrflg .eq. 1) WRITE (*,250) i,N1,n2,N3,N4,pthick,mat
          FORMAT (I5,4I6,F11.5,5X,I3)
          call diskroom (50)
          WRITE (2,250,err=2000) i,N1,n2,N3,N4,pthick,mat
      endif
      plteth(i)=entry(6)
      matcpl(i)=mat
      eyoung=youngm(mat)
      if ((pthick .ne. 0.) .and. (eyoung .ne. 0.)) then
          pratio=poisson(mat)
          indx=MAX(n1,n2,n3,n4)
          mxndif(n1)=MAX(mxndif(N1),indx)
          mxndif(n2)=MAX(mxndif(N2),indx)
          mxndif(n3)=MAX(mxndif(N3),indx)
          IF (n4 .GT. 0) mxndif(N4)=MAX(mxndif(N4),indx)
          diffnc(1,2)=coonod(1,N2)-coonod(1,N1)
          diffnc(2,2)=coonod(2,N2)-coonod(2,N1)
          diffnc(1,3)=coonod(1,N3)-coonod(1,N2)
          diffnc(2,3)=coonod(2,N3)-coonod(2,N2)
          IF (N4 .EQ. 0) THEN
              diffnc(1,1)=coonod(1,N1)-coonod(1,N3)
              diffnc(2,1)=coonod(2,N1)-coonod(2,N3)
          ELSE
              diffnc(1,4)=coonod(1,N4)-coonod(1,N3)
              diffnc(2,4)=coonod(2,N4)-coonod(2,N3)
              diffnc(1,1)=coonod(1,N1)-coonod(1,N4)
              diffnc(2,1)=coonod(2,N1)-coonod(2,N4)
          ENDIF
          INDX=1
          IF (diffnc(1,2)*diffnc(2,3) .GT. diffnc(2,2)*diffnc(1,3))
+              INDX=INDX+4
          IF (N4 .EQ. 0) THEN
              IF (INDX .EQ. 1) THEN
                  n=n2
                  n2=n3
                  n3=n
              ENDIF
          ELSE
+              IF (diffnc(1,3)*diffnc(2,4) .GT.
+                  diffnc(2,3)*diffnc(1,4)) INDX=INDX+2
+              IF (diffnc(1,4)*diffnc(2,1) .GT.
+                  diffnc(2,4)*diffnc(1,1)) INDX=INDX+1
          GOTO (260,270,280,300,310,280,300,320) indx
260      n=n2
          n2=n4
          n4=n
          GOTO 320
270      n=n2
          n2=n3
          n3=n
          GOTO 320
280      WRITE (*,290) i

```



```

290      FORMAT (' ERROR : ILLEGAL NODE DECLARATION FOR ',
+         'PLATE',I4,'.')
      call diskroom (50)
      WRITE (2,290,err=2000) i
      goto 994
300      n=n2
      n2=n3
      n3=n4
      n4=n
      GOTO 320
310      n=n3
      n3=n4
      n4=n
320      CONTINUE
ENDIF
nodepl(1,i)=N1
nodepl(2,i)=N2
nodepl(3,i)=N3
nodepl(4,i)=N4
IF (N4 .EQ. 0) THEN
  CALL triasemb (N1,N2,N3,pthick,eyoung,pratio)
ELSE
  coonod(1,nnodes+1)=(coonod(1,N1)+coonod(1,N2)+
+      coonod(1,N3)+coonod(1,N4))/4
  coonod(2,nnodes+1)=(coonod(2,N1)+coonod(2,N2)+
+      coonod(2,N3)+coonod(2,N4))/4
  CALL triasemb (N1,N2,nnodes+1,pthick,eyoung,pratio)
  CALL triasemb (N2,N3,nnodes+1,pthick,eyoung,pratio)
  CALL triasemb (N3,N4,nnodes+1,pthick,eyoung,pratio)
  CALL triasemb (N4,N1,nnodes+1,pthick,eyoung,pratio)
  ftcons(1)=stmqcn(1,1)*stmqcn(2,2)-
+      stmqcn(1,2)*stmqcn(2,1)
  invqcn(1,1)=stmqcn(2,2)/ftcons(1)
  invqcn(2,2)=stmqcn(1,1)/ftcons(1)
  invqcn(1,2)=-stmqcn(1,2)/ftcons(1)
  invqcn(2,1)=invqcn(1,2)
  DO 330 NI=1,4
  DO 330 MI=1,2
  n=(nodepl(NI,i)-1)*3+MI
  DO 330 NJ=NI,4
  IF (NJ .EQ. NI) THEN
    MK=MI
  ELSE
    MK=1
  ENDIF
  DO 330 mj=MK,2
  J=(nodepl(NJ,i)-1)*3+MJ
  k=min(n,j)
  l=max(n,j)-k+1
  longk=(malhbw+2)*(k-1)+l
  do 332 m=1,2
  longi=(malhbw+2)*(n-1)+malhbw+m
  ftcons(2)=0.
  do 331 mm=1,2
  longj=(malhbw+2)*(j-1)+malhbw+mm
331  ftcons(2)=ftcons(2)+stmtrx(longj)*invqcn(m,mm)
332  stmtrx(longk)=stmtrx(longk)-ftcons(2)*stmtrx(longi)
330  CONTINUE
  DO 350 NI=1,2
  DO 340 M=1,4
  DO 340 MI=1,2
  longi=(malhbw+2)*((nodepl(M,i)-1)*3+MI)+ni-2
  stmtrx(longi)=0.
340  CONTINUE
  DO 345 MI=1,2
345  stmqcn(mi,ni)=0.
350  CONTINUE
ENDIF
else
  if (scrflg .ge. 0) write (*,355) i
355  FORMAT (' WARNING : The plate',I4,
+      ' has been disconnected from the model.'/)
  call diskroom (70)
  WRITE (2,355,err=2000) i
  if ((echflg .eq. 0) .or. (scrflg .eq. 0)) write(*,217)

```

(continued)

BYTE LISTINGS SUPPLEMENT • JULY-SEPTEMBER 1986 39

40 BYTE LISTINGS SUPPLEMENT • JULY-SEPTEMBER, 1986


```

if (nextid(i,i1) .eq. 1) call pacer
longi=longi+malhbw+2
IF (igndof(i) .le. 0) then
  IF (ABS(stmtrx(longi)) .LT. .000001) THEN
    i1=(i-1)/3+1
    j1=i-3*(i1-1)
    WRITE (*,525) i1,j1
525    FORMAT (/' ERROR : THE STIFFNESS MATRIX APPEARS TO BE',
+           ' SINGULAR.'/' The elements connected to node ',i3
+           ' , do not contribute any stiffness in the free'/
+           ' degree of freedom ',i1,'.'/)
    call diskroom (162)
    WRITE (2,525,err=2000) i1,j1
    goto 994
  endif
DO 530 J=1,lenhbw(i)-1
  i=i+j
  IF ((igndof(i) .le. 0) .and. (stmtrx(longi+j) .ne. 0.)) then
    RATIO=stmtrx(longi+j)/stmtrx(longi)
    longl=(malhbw+2)*j+longi-1
    DO 529 k=j+1,lenhbw(i)
      longl=longl+1
      IF (igndof(i) .le. 0) stmtrx(longl)=stmtrx(longl)-
529        ratio*stmtrx(longi-1+k)
    CONTINUE
    stmtrx(longi+j)=ratio
    disdof(i)=disdof(i)-RATIO*disdof(i)
  endif
530  CONTINUE
  disdof(i)=disdof(i)/stmtrx(longi)
ENDIF
535 CONTINUE
write (*,536)
536 format (/' PASS 2 : BACKWARDS SUBSTITUTION')
write (*,524) arrow,(dash,i=1,j1),arrow,(blank,j=1,k1)
if (nextid(numdof,i1) .eq. 1) call pacer
DO 550 i=numdof-1,1,-1
  if (nextid(i,i1) .eq. 1) call pacer
  longj=(malhbw+2)*(i-1)+1
  IF (igndof(i) .le. 0) then
    DO 540 K=1,lenhbw(i)-1
      disdof(i)=disdof(i)-stmtrx(longj+k)*disdof(i+k)
540    endif
  550 CONTINUE
  write (*,555)
  555 format (/' The system has been succesfully solved.')
```

C
C Print displacements
C

```

  if (scrflg .eq. 0) then
    diamsq='Writing results to file
+
    call setstr (110,diamsq)
    call setstr (78,outfil)
    call movstr (diamsq,25,1,outfil,1,77)
    call resstr (outfil)
    call pakstr (diamsq)
    call constr (diamsq,elipss)
    call expstr (diamsq)
    call resstr (diamsq)
    write (*,680) diamsq
680    format (/1X,A/' Displacements...\')
  endif
  if (scrflg .eq. 1) THEN
    WRITE (*,685)
685    FORMAT (/' NODE DISPLACEMENTS'//
+           ' Node      U      V      Omega'/)
  endif
  call diskroom (76)
  WRITE (2,685,err=2000)
  DO 700 J=1,nnodes
    if (mxndif(j) .ne. 0) then
      if (scrflg .eq. 1) WRITE (*,690) j,(disdof(3*(j-1)+i),i=1,3)
690    FORMAT (15,1X,3F12.6)
      call diskroom (44)
      WRITE (2,690,err=2000) j,(disdof(3*(j-1)+i),i=1,3)

```

(continued)

```

endif
700 CONTINUE
C
C Beam corner forces
C
  IF (nbeams .gt. 0) then
    if (scrflg .eq. 1) then
      WRITE (*,710)
      FORMAT (// ' BEAM CORNER FORCES' //
        + ' Beam      I      J      FX1      FY1      MZ1' ,
        + '      FX2      FY2      MZ2' //)
    else
      if (scrflg .eq. 0) write (*,205)
    endif
    call diskroom (125)
    WRITE (2,710,err=2000)
    DO 740 i=1,nbeams
      mat=matcbm(i)
      eyoung=youngm(mat)
      if ((eyoung .ne. 0.) .and. (bmarea(i) .ne. 0.)) then
        n1=nodebm(1,i)
        n2=nodebm(2,i)
        diffnc(1,2)=coonod(1,n2)-coonod(1,n1)
        diffnc(2,2)=coonod(2,n2)-coonod(2,n1)
        blngth=DSQRT(diffnc(1,2)*diffnc(1,2)+
          + diffnc(2,2)*diffnc(2,2))
        bmlcos=diffnc(1,2)/blngth
        bmlsin=diffnc(2,2)/blngth
        I1=3*n1-2
        J1=3*n2-2
        ftcons(1)=disdof(J1)-disdof(I1)
        ftcons(2)=disdof(J1+1)-disdof(I1+1)
        ftcons(3)=3*(bmlsin*ftcons(1)-bmlcos*ftcons(2))/blngth
        ftcons(4)=(bmdis1(I1)+bmdis2(I1))*blngth/2.
        ftcons(5)=2*eyoung*bminer(I1)/blngth
        ftcons(6)=eyoung*bmarea(I1)*(bmlcos*ftcons(1)+
          + bmlsin*ftcons(2))/blngth
        beamcf(3,1)=ftcons(5)*(ftcons(3)+2*disdof(I1+2)+
          + disdof(J1+2))-(8.*ftcons(4)-bmdis2(I1)*
          + blngth/2.)*blngth/90.
        beamcf(3,2)=ftcons(5)*(2*disdof(J1+2)+disdof(I1+2)+
          + ftcons(3))+(8.*ftcons(4)-bmdis1(I1)*
          + blngth/2.)*blngth/90.
        ftcons(7)=(ftcons(4)+bmdis1(I1)*blngth/2.)/3.-
          + (beamcf(3,1)+beamcf(3,2))/blngth
        ftcons(8)=ftcons(7)-ftcons(4)
        beamcf(1,1)=-bmlcos*ftcons(6)+bmlsin*ftcons(7)
        beamcf(1,2)=bmlcos*ftcons(6)-bmlsin*ftcons(7)
        beamcf(2,1)=-bmlsin*ftcons(6)-bmlcos*ftcons(7)
        beamcf(2,2)=bmlsin*ftcons(6)+bmlcos*ftcons(7)
        DO 720 j=1,2
        DO 720 k=1,3
        reafor(k,nodebm(j,i))=reafor(k,nodebm(j,i))+beamcf(k,j)
720      CONTINUE
        baxial(i)=ftcons(6)
        bshear(1,i)=ftcons(7)
        bshear(2,i)=ftcons(8)
        bmomnt(1,i)=-beamcf(3,1)
        bmomnt(2,i)=beamcf(3,2)
        if (scrflg .eq. 1) WRITE (*,730) i,n1,n2,
          + (beamcf(k,1),k=1,3),(beamcf(k,2),k=1,3)
730      FORMAT (I5,2I6,1X,6F12.0)
        call diskroom (92)
        WRITE (2,730,err=2000) i,n1,n2,(beamcf(k,1),k=1,3),
          + (beamcf(k,2),k=1,3)
      endif
740    CONTINUE
C
C Beam loads and stresses
C
    if (scrflg .eq. 1) WRITE (*,750)
    FORMAT (// ' BEAM LOADS AND STRESSES' //
      + ' Beam      I      J      PAX      SAX      ' ,
      + ' SH1      SH2      BM1      BM2' //)
    call diskroom (130)
    WRITE (2,750,err=2000)

```



```

DO 760 i=1,nbeams
mat=matcbm(i)
if ((youngm(mat) .ne. 0.) .and. (bmarea(i) .ne. 0.)) then
    ftcons(1)=baxial(i)/bmarea(i)
    if (scrflg .eq. 1) WRITE (*,730) i,(nodebm(k,i),k=1,2),
+       baxial(i),ftcons(1),(bshear(k,i),k=1,2),(bmomnt(k,i),k=1,2)
    call diskroom (92)
    WRITE (2,730,err=2000) i,(nodebm(k,i),k=1,2),baxial(i),
+       ftcons(1),(bshear(k,i),k=1,2),(bmomnt(k,i),k=1,2)
    endif
760    continue
endif

C
C Plate corner forces
C
IF (nplate .gt. 0) then
    if (scrflg .eq. 1) then
770        WRITE (*,770)
        FORMAT (// ' PLATE CORNER FORCES' //
+             ' Plate      I      J      K      L      FX1      FY1      ',
+             'FX2      FY2      FX3      FY3      FX4      FY4' //)
    else
        if (scrflg .eq. 0) write (*,245)
    endif
    call diskroom (138)
    WRITE (2,770,err=2000)
    DO 850 LPL=1,nplate
    TH=plteht(LPL)
    mat=matcpl(lpl)
    eyoung=youngm(mat)
    pratio=poisson(mat)
    if ((th .ne. 0.) .and. (eyoung .ne. 0.)) then
        DO 780 I=1,2
        DO 780 J=1,4
        pltecf(I,J)=0.
780        IF (nodepl(4,LPL) .EQ. 0) THEN
            CALL triloads (1,2,3,th,eyoung,pratio,lpl,nodepl)
        ELSE
            coonod(1,nnodes+1)=(coonod(1,nodepl(1,LPL))+
+                coonod(1,nodepl(2,LPL))+coonod(1,nodepl(3,LPL))+
+                coonod(1,nodepl(4,LPL)))/4
            coonod(2,nnodes+1)=(coonod(2,nodepl(1,LPL))+
+                coonod(2,nodepl(2,LPL))+coonod(2,nodepl(3,LPL))+
+                coonod(2,nodepl(4,LPL)))/4
            ftcons(7)=0
            ftcons(8)=0
            ftcons(9)=0
            DO 790 i=1,8
            DO 790 J=1,2
790            sttemp(i,j)=0.
            Inp(3)=nnodes+1
            DO 810 I=1,4
            J=nextid(I,4)
            Inp(1)=nodepl(I,LPL)
            Inp(2)=nodepl(J,LPL)
            DO 800 N1=1,2
            DO 800 N2=1,3
800            diffnc(N1,N2)=coonod(N1,inp(N2))-
+                coonod(N1,inp(previ(N2,3)))
            ftcons(1)=diffnc(2,3)*diffnc(1,2)-
+                diffnc(1,3)*diffnc(2,2)
            ftcons(2)=1/(ftcons(1)*(1+pratio))
            ftcons(3)=ftcons(2)*(diffnc(1,3)*diffnc(1,2)+
+                diffnc(2,3)*diffnc(2,2))
            ftcons(4)=ftcons(2)*(diffnc(2,3)*diffnc(1,2)-
+                diffnc(1,3)*diffnc(2,2))
            ftcons(5)=ftcons(2)*(diffnc(1,2)*diffnc(1,2)+
+                diffnc(2,2)*diffnc(2,2))
            ftcons(6)=1/(ftcons(1)*(1-pratio))
            sttemp(2*i-1,1)=sttemp(2*I-1,1)+ftcons(3)+
+                ftcons(6)*diffnc(2,2)*diffnc(2,3)
            sttemp(2*i-1,2)=sttemp(2*I-1,2)+ftcons(4)-
+                ftcons(6)*diffnc(2,3)*diffnc(1,2)
            sttemp(2*i,1)=sttemp(2*I,1)-ftcons(4)-
+                ftcons(6)*diffnc(2,2)*diffnc(1,3)

```

(continued)

```

      sttemp(2*i,2)=sttemp(2*I,2)+ftcons(3)+
+      ftcons(6)*diffnc(1,3)*diffnc(1,2)
      sttemp(2*j-1,1)=sttemp(2*j-1,1)-ftcons(3)-ftcons(5)+
+      ftcons(6)*diffnc(2,2)*diffnc(2,1)
      sttemp(2*j-1,2)=sttemp(2*j-1,2)-ftcons(4)-
+      ftcons(6)*diffnc(2,1)*diffnc(1,2)
      sttemp(2*j,1)=sttemp(2*j,1)+ftcons(4)-
+      ftcons(6)*diffnc(2,2)*diffnc(1,1)
      sttemp(2*j,2)=sttemp(2*j,2)-ftcons(3)-ftcons(5)+
+      ftcons(6)*diffnc(1,1)*diffnc(1,2)
      ftcons(7)=ftcons(7)+ftcons(5)+
+      ftcons(6)*diffnc(2,2)*diffnc(2,2)
      ftcons(8)=ftcons(8)-
+      ftcons(6)*diffnc(1,2)*diffnc(2,2)
      ftcons(9)=ftcons(9)+ftcons(5)+
+      ftcons(6)*diffnc(1,2)*diffnc(1,2)
810  CONTINUE
      ftcons(1)=0
      ftcons(2)=0
      DO 820 I=1,4
      ftcons(1)=ftcons(1)-
+      sttemp(2*I-1,1)*disdof(3*nodepl(I,LPL)-2)-
+      sttemp(2*I,1)*disdof(3*nodepl(I,LPL)-1)
      ftcons(2)=ftcons(2)-
+      sttemp(2*I-1,2)*disdof(3*nodepl(I,LPL)-2)-
+      sttemp(2*I,2)*disdof(3*nodepl(I,LPL)-1)
820  CONTINUE
      ftcons(3)=ftcons(7)*ftcons(9)-ftcons(8)*ftcons(8)
      disdof(numdof+1)=(ftcons(1)*ftcons(9)-
+      ftcons(8)*ftcons(2))/ftcons(3)
      disdof(numdof+2)=(ftcons(2)*ftcons(7)-
+      ftcons(8)*ftcons(1))/ftcons(3)
      i=-nnodes-1
      CALL triloads (1,2,i,th,eyoung,pratio,lpl,nodepl)
      CALL triloads (2,3,i,th,eyoung,pratio,lpl,nodepl)
      CALL triloads (3,4,i,th,eyoung,pratio,lpl,nodepl)
      CALL triloads (4,1,i,th,eyoung,pratio,lpl,nodepl)
      DO 830 I=1,3
830  plstrs(I,LPL)=plstrs(I,LPL)/4
      ENDIF
      if (scrflg .eq. 1) WRITE (*,840) LPL,(nodepl(k,LPL),k=1,4)
+      ,((pltecf(i,j),i=1,2),j=1,4)
840  FORMAT (I5,4I6,1X,8F9.0)
      call diskroom (104)
      WRITE (2,840,err=2000) LPL,(nodepl(k,LPL),k=1,4),
+      ,((pltecf(i,j),i=1,2),j=1,4)
      endif
850  CONTINUE
C
C Plate load-intensities and stresses
C
      if (scrflg .eq. 1) WRITE (*,860)
860  FORMAT (//' PLATE LOAD INTENSITIES AND STRESSES'//
+      ' Plate   I       J       K       L       PIX       PIY       TXY',
+      '      SX       SY       TAU       SMAX       SMIN       TMAX',
+      '      Angle'//)
      call diskroom (172)
      WRITE (2,860,err=2000)
      DO 890 LPL=1,nplate
      mat=matcpl(lpl)
      if ((plteth(lpl) .ne. 0.) .and. (youngm(mat) .ne. 0.)) then
      DO 870 I=1,3
870  plints(I)=plstrs(I,LPL)*plteth(LPL)
      ftcons(3)=SQRT(plstrs(3,LPL)*plstrs(3,LPL)+
+      .25*(plstrs(2,LPL)-plstrs(1,LPL))*(plstrs(2,LPL)-
+      plstrs(1,LPL)))
      ftcons(5)=.5*(plstrs(1,LPL)+plstrs(2,LPL))
      ftcons(1)=ftcons(5)+ftcons(3)
      ftcons(2)=ftcons(5)-ftcons(3)
      ftcons(4)=degree(2*plstrs(3,LPL),
+      plstrs(2,LPL)-plstrs(1,LPL))/2.
      if (scrflg .eq. 1) WRITE (*,880) LPL,(nodepl(k,LPL),k=1,4)
+      , (plints(k),k=1,3), (plstrs(k,LPL),k=1,3)
+      , (ftcons(k),k=1,4)
880  FORMAT (I5,4I6,1X,10F9.0)
      call diskroom (122)

```



```

      WRITE (2,880,err=2000) LPL,(nodepl(k,LPL),k=1,4),
+      (plints(k),k=1,3),(plstrs(k,LPL),k=1,3),(ftcons(k),k=1,4)
      endif
890    CONTINUE
C
C Plate stresses at node points
C
      if (scrflg .eq. 1) WRITE (*,900)
900    FORMAT (//' PLATE STRESSES AT NODE POINTS'//
+      ' Node   Coordinate X   Coordinate Y   SX   SY   ',
+      ' TAU   SMAX   SMIN   TMAX   Angle'//)
      call diskroom (151)
      WRITE (2,900,err=2000)
      DO 930 I=1,nnodes
      k=0
      DO 910 J=1,3
      IF (pstnor(J,I) .GT. 0.) THEN
        ftcons(J)=pstacc(J,I)/pstnor(J,I)
      ELSE
        k=1
      ENDIF
910    CONTINUE
      if (k .ne. 1) then
+      ftcons(6)=SQRT(ftcons(3)*ftcons(3)+
+      .25*(ftcons(2)-ftcons(1))*(ftcons(2)-ftcons(1)))
+      ftcons(8)=.5*(ftcons(1)+ftcons(2))
+      ftcons(4)=ftcons(8)+ftcons(6)
+      ftcons(5)=ftcons(8)-ftcons(6)
+      ftcons(7)=degree(sngl(2*ftcons(3)),
+      sngl(ftcons(2)-ftcons(1)))/2.
+      if (scrflg .eq. 1) WRITE (*,920) I,coonod(1,I),coonod(2,I),
+      (ftcons(k),k=1,7)
920    FORMAT (I5,3X,F12.5,3X,F12.5,7F10.0)
      call diskroom (107)
      WRITE (2,920,err=2000) I,coonod(1,I),coonod(2,I),
+      (ftcons(k),k=1,7)
      endif
930    CONTINUE
      endif
C
C Fastener forces and stresses
C
      IF (nfastn .gt. 0) then
        if (scrflg .eq. 1) then
          WRITE (*,940)
940        FORMAT (//' FASTENER FORCES AND STRESSES'//
+        ' Fastener I   J   FX   FY   F   ',
+        ' Angle   Stress'//)
        else
          if (scrflg .eq. 0) write (*,385)
        endif
        call diskroom (113)
        WRITE (2,940,err=2000)
        DO 960 LFS=1,nfastn
          if (fsstif(lfs) .ne. 0.) then
            n1=nodefs(1,LFS)
            n2=nodefs(2,LFS)
            I1=3*n1-2
            J1=3*n2-2
            ftcons(1)=fsstif(lfs)*(disdof(I1)-disdof(J1))
            ftcons(2)=fsstif(lfs)*(disdof(I1+1)-disdof(J1+1))
            ftcons(3)=SQRT(ftcons(1)*ftcons(1)+ftcons(2)*ftcons(2))
            ftcons(4)=degree(sngl(ftcons(2)),sngl(ftcons(1)))
            ftcons(5)=ftcons(3)/fsarea(lfs)
            if (scrflg .eq. 1) WRITE (*,950) LFS,n1,n2,
+            (ftcons(k),k=1,5)
950          FORMAT (I5,2I6,1X,5F10.0)
          call diskroom (70)
          WRITE (2,950,err=2000) LFS,n1,n2,(ftcons(k),k=1,5)
          reafor(1,n1)=reafor(1,n1)+ftcons(1)
          reafor(1,n2)=reafor(1,n2)-ftcons(1)
          reafor(2,n1)=reafor(2,n1)+ftcons(2)
          reafor(2,n2)=reafor(2,n2)-ftcons(2)
        endif
960      CONTINUE
      endif

```

(continued)

July

```

C
C Node internal forces and reactions
C
      reac1b(1)='      '
      reac1b(2)='      '
      reac1b(3)='Reaction'
      if (scrflg .eq. 1) then
        WRITE (*,970)
970      FORMAT (// ' NODE INTERNAL FORCES AND REACTIONS'//
+      ' Node      Coordinate X      Coordinate Y      FX',
+      '                      FY                      MZ'//)
      else
        if (scrflg .eq. 0) write (*,972)
972      format ('Reactions...\')
      endif
      call diskroom (142)
      WRITE (2,970,err=2000)
      DO 990 I=1,nnodes
        if (scrflg .eq. 1) WRITE (*,980) I,coonod(1,I),coonod(2,I),
+      (reafor(I,I),reac1b(1+abs(igndof((i-1)*3+j))),j=1,3)
980      FORMAT (I5,3X,F12.5,3X,F12.5,3(F12.0,1X,a8,1X))
        call diskroom (103)
        WRITE (2,980,err=2000) I,coonod(1,I),coonod(2,I),
+      (reafor(I,I),reac1b(1+abs(igndof((i-1)*3+j))),j=1,3)
990      CONTINUE
        if (scrflg .eq. 0) write (*,512)
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C                                REPORT THE EXECUTION TIME                                C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C Report the execution time
C
994      cpusec=0.
      call time (lasthr,lastmn,lastsc,lasths)
      if (lasthr .lt. inithr) cpusec=86400.
      cpusec=cpusec+3600.*(lasthr-inithr)+60.*(lastmn-initmn)+lastsc-
+      initsec+.01*(lasths-iniths)
      if (scrflg .ge. 0) write (*,995) cpusec
995      format (// ' Execution time : ',f8.2,' seconds.')
      if (ierror .ne. -1) then
        call diskroom (43)
        write (2,995,err=2000) cpusec
      endif
      write (*,999)
999      format (' ')
      STOP
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C                                REPORT UNSPECIFIED I/O ERRORS DETECTED                                C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
1000      write (*,1010)
1010      format (// ' ERROR : CANNOT READ INPUT FILE.'/
+      ' The program cannot continue.')
      goto 994
2000      write (*,2010)
2010      format (// ' ERROR : CANNOT WRITE OUTPUT FILE.'/
+      ' The program cannot continue.')
      ierror=-1
      goto 994
      END

```

expstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE EXPSTR - SUBROUTINE TO EXPAND A STRING TO ITS SIZE
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

EXPSTR is a routine designed to be called from FORTRAN as a subroutine to expand a string from its actual length to its size by filling the string with blanks.

Mode of use:

call EXPSTR (string)

where

string = name of the string (variable of type CHARACTER) to be expanded.

*

SUBTTL FORMAL DECLARATIONS

PAGE

```
csexps  SEGMENT 'CODE'
        ASSUME  CS:csexps
```

SUBTTL EXPSTR - EXECUTABLE CODE

PAGE

```
PUBLIC  expstr
expstr PROC  FAR
```

```
PUSH    BP
MOV     BP,SP
PUSH    ds
LDS     BX,DWORD PTR [BP+6]
mov     ah,32
```

; Scan for either end-of-string mark.

scanend:

```
mov     al,[bx]
cmp     al,0
jz      exit
cmp     al,6
jz      fillblanks
inc     bx
jmp     scanend
```

; Fill with blanks between logical and physical end

fillblanks:

```
MOV     [BX],ah
inc     bx
mov     al,[bx]
cmp     al,0
jnz     fillblanks
```

; Exit

exit:

```
POP     ds
MOV     SP,BP
POP     BP
RET     4
```

```
expstr ENDP
csexps ENDS
```

END

punch.inp

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

This file demonstrates the mixture of text (notes) and input data allowed in the Microsafe Program Series. The data for the program is contained on the lines that contain a slash character. All lines without a slash and any characters following the slash are ignored by the program.

```
15 / # of nodes
1 / # of materials
4 / # of beams
8 / # of plates
0 / # of fasteners
```

(continued)

```

1 / # of loaded nodes
11 / # of imposed restraints
1 0.0 10.0 / node-# x-coordinate y-coordinate
2 0.0 0.0 /
3 0.0 -10.0 /
4 10.0 10.0 /
5 10.0 0.0 /
6 10.0 -10.0 /
7 20.0 10.0 /
8 20.0 0.0 /
9 20.0 -10.0 /
10 30.0 10.0 /
11 30.0 0.0 /
12 30.0 -10.0 /
13 40.0 10.0 /
14 40.0 0.0 /
15 40.0 -10.0 /
1 10.3E6 0.33 / material-# Young's-modulus Poisson's-ratio
1 2 5 0.5 0.1 1 0.0 0.0 / beam-# node1 node2 area inertia material-# Q1 Q2
2 5 8 0.5 0.1 1 0.0 0.0 /
3 8 11 0.5 0.1 1 0.0 0.0 /
4 11 14 0.5 0.1 1 0.0 0.0 /
1 2 5 4 1 0.06 1 / plate-# node1 node2 node3 node4 thickness material-#
2 5 8 7 4 0.06 1 /
3 8 11 10 7 0.06 1 /
4 11 14 13 10 0.06 1 /
5 3 6 5 2 0.06 1 /
6 6 9 8 5 0.06 1 /
7 9 12 11 8 0.06 1 /
8 12 15 14 11 0.06 1 /
2 10000. 0.0 0.0 / node # load-x load-y moment
1 1 0.0 / node-# restrained-freedom-# imposed-deflection
3 1 0.0 /
4 1 0.0 /
6 1 0.0 /
7 1 0.0 /
9 1 0.0 /
10 1 0.0 /
12 1 0.0 /
13 1 0.0 /
14 2 0.0 / never remove this node - used to eliminate freebody translation
15 1 0.0 /

```

```

1 -> restrain x-direction
2 -> restrain y-direction
3 -> restrain rotation

```

lug.inp

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

File: LUG.INP is the idealization of a lug with a bolt in the center and bolted to a strongback at each end of the flange. Load angle is 0 deg.

```

247 / NUMBER OF NODES
1 / NUMBER OF MATERIALS
24 / NUMBER OF BEAMS
222 / NUMBER OF PLATES
0 / NUMBER OF FASTENERS
11 / NUMBER OF LOADED NODES
20 / NUMBER OF RESTRAINED NODES
169 0.0000 1.3000 / NODAL DEFINITION
188 0.0000 1.8000 /
187 -0.1294 1.7829 /
186 -0.2500 1.7330 /
185 -0.3535 1.6535 /
184 -0.4330 1.5500 /
183 -0.4829 1.4294 /
182 -0.5000 1.3000 /
168 -0.4829 1.1706 /
144 -0.4330 1.0500 /
145 -0.3535 0.9464 /
146 -0.2500 0.8670 /
147 -0.1294 0.8171 /
148 0.0000 0.8000 /

```


149	0.1294	0.8171	/
150	0.2499	0.8670	/
151	0.3535	0.9464	/
152	0.4330	1.0500	/
170	0.4829	1.1706	/
194	0.5000	1.3000	/
193	0.4829	1.4294	/
192	0.4330	1.5499	/
191	0.3535	1.6535	/
190	0.2500	1.7330	/
189	0.1294	1.7829	/
209	0.0000	1.9000	/
208	-0.1552	1.8795	/
207	-0.3000	1.8196	/
206	-0.4242	1.7242	/
205	-0.5196	1.6000	/
204	-0.5795	1.4552	/
181	-0.6000	1.3000	/
167	-0.5795	1.1448	/
142	-0.5196	1.0000	/
143	-0.4242	0.8757	/
121	-0.3000	0.7804	/
122	-0.1552	0.7205	/
123	0.0000	0.7000	/
124	0.1552	0.7205	/
125	0.2999	0.7804	/
153	0.4242	0.8757	/
154	0.5196	1.0000	/
171	0.5795	1.1448	/
195	0.6000	1.3000	/
214	0.5795	1.4552	/
213	0.5196	1.5999	/
212	0.4242	1.7242	/
211	0.3000	1.8196	/
210	0.1552	1.8795	/
226	0.0000	2.0000	/
225	-0.1811	1.9761	/
224	-0.3500	1.9062	/
223	-0.4949	1.7949	/
222	-0.6062	1.6500	/
203	-0.6761	1.4811	/
180	-0.7000	1.3000	/
166	-0.6761	1.1189	/
141	-0.6062	0.9500	/
120	-0.4949	0.8051	/
119	-0.3500	0.6938	/
101	-0.1811	0.6239	/
102	0.0000	0.6000	/
103	0.1811	0.6239	/
126	0.3499	0.6938	/
127	0.4949	0.8051	/
155	0.6062	0.9499	/
172	0.6761	1.1189	/
196	0.7000	1.3000	/
215	0.6761	1.4811	/
230	0.6062	1.6499	/
229	0.4949	1.7949	/
228	0.3500	1.9062	/
227	0.1811	1.9761	/
238	0.0000	2.1000	/
237	-0.2070	2.0727	/
236	-0.4000	1.9928	/
235	-0.5656	1.8656	/
221	-0.6928	1.7000	/
202	-0.7727	1.5070	/
179	-0.8000	1.3000	/
165	-0.7727	1.0930	/
140	-0.6928	0.9001	/
118	-0.5656	0.7344	/
100	-0.4000	0.6072	/
80	-0.2070	0.5273	/
81	0.0000	0.5000	/
82	0.2070	0.5273	/
104	0.3999	0.6072	/
128	0.5656	0.7344	/

(continued)

July

156	0.6928	0.9000	/
173	0.7727	1.0930	/
197	0.8000	1.3000	/
216	0.7727	1.5070	/
231	0.6928	1.6999	/
241	0.5656	1.8656	/
240	0.4000	1.9928	/
239	0.2070	2.0727	/
245	0.0000	2.2000	/
244	-0.2329	2.1693	/
243	-0.4500	2.0794	/
234	-0.6363	1.9363	/
220	-0.8000	1.7650	/
219	-0.9400	1.6250	/
200	-1.0900	1.4600	/
201	-1.0000	1.4600	/
178	-1.0000	1.3000	/
164	-1.0000	1.1400	/
139	-0.8000	0.9800	/
138	-1.0000	0.9800	/
117	-0.8000	0.8200	/
99	-0.6000	0.6350	/
79	-0.4000	0.5000	/
83	0.4000	0.5000	/
105	0.6000	0.6350	/
129	0.8000	0.8200	/
158	1.0000	0.9800	/
157	0.8000	0.9800	/
174	1.0000	1.1400	/
198	1.0000	1.3000	/
217	1.0000	1.4600	/
218	1.0900	1.4600	/
233	0.9400	1.6250	/
232	0.8000	1.7650	/
242	0.6363	1.9363	/
247	0.4500	2.0794	/
246	0.2329	2.1690	/
177	-1.2400	1.3000	/
162	-1.4000	1.1400	/
163	-1.2000	1.1400	/
137	-1.2000	0.9800	/
116	-1.0000	0.8200	/
98	-0.8000	0.6600	/
78	-0.6000	0.5000	/
56	-0.4000	0.3400	/
57	-0.2000	0.3400	/
58	0.0000	0.3400	/
59	0.2000	0.3400	/
60	0.4000	0.3400	/
84	0.6000	0.5000	/
106	0.8000	0.6600	/
130	1.0000	0.8200	/
159	1.2000	0.9800	/
175	1.2000	1.1400	/
176	1.4000	1.1400	/
199	1.2400	1.3000	/
135	-1.5450	0.9800	/
112	-1.6950	0.8200	/
113	-1.6000	0.8200	/
136	-1.4000	0.9800	/
114	-1.4000	0.8200	/
95	-1.4000	0.6600	/
115	-1.2000	0.8200	/
96	-1.2000	0.6600	/
97	-1.0000	0.6600	/
76	-1.0000	0.5000	/
77	-0.8000	0.5000	/
54	-0.8000	0.3400	/
55	-0.6000	0.3400	/
32	-0.6000	0.1700	/
33	-0.4000	0.1700	/
34	-0.2000	0.1700	/
35	0.0000	0.1700	/
36	0.2000	0.1700	/
37	0.4000	0.1700	/
61	0.6000	0.3400	/

62	0.8000	0.3400	/
85	0.8000	0.5000	/
86	1.0000	0.5000	/
107	1.0000	0.6600	/
108	1.2000	0.6600	/
131	1.2000	0.8200	/
132	1.4000	0.8200	/
133	1.6000	0.8200	/
134	1.6950	0.8200	/
160	1.4000	0.9800	/
161	1.5450	0.9800	/
93	-1.8450	0.6600	/
94	-1.6000	0.6600	/
71	-2.0000	0.5000	/
72	-1.8000	0.5000	/
73	-1.6000	0.5000	/
74	-1.4000	0.5000	/
75	-1.2000	0.5000	/
50	-1.6000	0.3400	/
51	-1.4000	0.3400	/
52	-1.2000	0.3400	/
53	-1.0000	0.3400	/
29	-1.2000	0.1700	/
30	-1.0000	0.1700	/
31	-0.8000	0.1700	/
8	-0.8000	0.0000	/
9	-0.6000	0.0000	/
10	-0.4000	0.0000	/
11	-0.2000	0.0000	/
12	0.0000	0.0000	/
13	0.2000	0.0000	/
14	0.4000	0.0000	/
15	0.6000	0.0000	/
16	0.8000	0.0000	/
38	0.6000	0.1700	/
39	0.8000	0.1700	/
63	1.0000	0.3400	/
64	1.2000	0.3400	/
65	1.4000	0.3400	/
66	1.6000	0.3400	/
87	1.2000	0.5000	/
88	1.4000	0.5000	/
89	1.6000	0.5000	/
90	1.8000	0.5000	/
91	2.0000	0.5000	/
109	1.4000	0.6600	/
110	1.6000	0.6600	/
111	1.8450	0.6600	/
48	-2.0000	0.3400	/
49	-1.8000	0.3400	/
25	-2.0000	0.1700	/
26	-1.8000	0.1700	/
27	-1.6000	0.1700	/
28	-1.4000	0.1700	/
40	1.0000	0.1700	/
41	1.2000	0.1700	/
42	1.4000	0.1700	/
43	1.6000	0.1700	/
44	1.8000	0.1700	/
45	2.0000	0.1700	/
67	1.8000	0.3400	/
68	2.0000	0.3400	/
2	-2.0000	0.0000	/
3	-1.8000	0.0000	/
4	-1.6000	0.0000	/
5	-1.4000	0.0000	/
6	-1.2000	0.0000	/
7	-1.0000	0.0000	/
17	1.0000	0.0000	/
18	1.2000	0.0000	/
19	1.4000	0.0000	/
20	1.6000	0.0000	/
21	1.8000	0.0000	/
22	2.0000	0.0000	/
70	-2.2000	0.5000	/

(continued)

July

120	97	98	117	116	0.750	1 /
121	71	72	93	0	0.450	1 /
122	72	73	94	93	0.450	1 /
123	73	74	95	94	0.450	1 /
124	74	75	96	95	0.450	1 /
125	75	76	97	96	0.600	1 /
126	76	77	98	97	0.750	1 /
127	77	78	99	98	0.750	1 /
128	78	79	100	99	0.750	1 /
129	198	199	218	217	0.450	1 /
130	174	175	199	198	0.450	1 /
131	175	176	199	0	0.250	1 /
132	158	159	175	174	0.450	1 /
133	159	160	176	175	0.250	1 /
134	160	161	176	0	0.250	1 /
135	129	130	158	157	0.750	1 /
136	130	131	159	158	0.450	1 /
137	131	132	160	159	0.250	1 /
138	132	133	161	160	0.250	1 /
139	133	134	161	0	0.250	1 /
140	106	107	130	129	0.750	1 /
141	107	108	131	130	0.450	1 /
142	108	109	132	131	0.250	1 /
143	109	110	133	132	0.250	1 /
144	110	111	134	133	0.250	1 /
145	83	84	105	104	0.750	1 /
146	84	85	106	105	0.750	1 /
147	85	86	107	106	0.750	1 /
148	86	87	108	107	0.600	1 /
149	87	88	109	108	0.450	1 /
150	88	89	110	109	0.450	1 /
151	89	90	111	110	0.450	1 /
152	90	91	111	0	0.450	1 /
153	47	48	71	70	0.550	1 /
154	48	49	72	71	0.550	1 /
155	49	50	73	72	0.650	1 /
156	50	51	74	73	0.750	1 /
157	51	52	75	74	0.750	1 /
158	52	53	76	75	0.750	1 /
159	53	54	77	76	0.750	1 /
160	54	55	78	77	0.750	1 /
161	55	56	79	78	0.750	1 /
162	56	57	80	79	0.750	1 /
163	57	58	81	80	0.750	1 /
164	58	59	82	81	0.750	1 /
165	59	60	83	82	0.750	1 /
166	60	61	84	83	0.750	1 /
167	61	62	85	84	0.750	1 /
168	62	63	86	85	0.750	1 /
169	63	64	87	86	0.750	1 /
170	64	65	88	87	0.750	1 /
171	65	66	89	88	0.750	1 /
172	66	67	90	89	0.650	1 /
173	67	68	91	90	0.550	1 /
174	68	69	92	91	0.550	1 /
175	24	25	48	47	0.550	1 /
176	25	26	49	48	0.550	1 /
177	26	27	50	49	0.650	1 /
178	27	28	51	50	0.750	1 /
179	28	29	52	51	0.750	1 /
180	29	30	53	52	0.750	1 /
181	30	31	54	53	0.750	1 /
182	31	32	55	54	0.750	1 /
183	32	33	56	55	0.750	1 /
184	33	34	57	56	0.750	1 /
185	34	35	58	57	0.750	1 /
186	35	36	59	58	0.750	1 /
187	36	37	60	59	0.750	1 /
188	37	38	61	60	0.750	1 /
189	38	39	62	61	0.750	1 /
190	39	40	63	62	0.750	1 /
191	40	41	64	63	0.750	1 /
192	41	42	65	64	0.750	1 /
193	42	43	66	65	0.750	1 /
194	43	44	67	66	0.650	1 /
195	44	45	68	67	0.550	1 /


```

196 45 46 69 68 0.550 1 /
197 1 2 25 24 0.550 1 /
198 2 3 26 25 0.550 1 /
199 3 4 27 26 0.650 1 /
200 4 5 28 27 0.750 1 /
201 5 6 29 28 0.750 1 /
202 6 7 30 29 0.750 1 /
203 7 8 31 30 0.750 1 /
204 8 9 32 31 0.750 1 /
205 9 10 33 32 0.750 1 /
206 10 11 34 33 0.750 1 /
207 11 12 35 34 0.750 1 /
208 12 13 36 35 0.750 1 /
209 13 14 37 36 0.750 1 /
210 14 15 38 37 0.750 1 /
211 15 16 39 38 0.750 1 /
212 16 17 40 39 0.750 1 /
213 17 18 41 40 0.750 1 /
214 18 19 42 41 0.750 1 /
215 19 20 43 42 0.750 1 /
216 20 21 44 43 0.650 1 /
217 21 22 45 44 0.550 1 /
218 22 23 46 45 0.550 1 /
219 117 99 118 0 0.750 1 /
220 105 129 128 0 0.750 1 /
221 100 80 101 0 0.750 1 /
222 82 104 103 0 0.750 1 /
169 0.0 25000.0 0 / NODAL LOADS - N1 PX PY MZ
2 0.0 5000.0 0 /
3 0.0 5000.0 0 /
4 0.0 5000.0 0 /
5 0.0 5000.0 0 /
19 0.0 2500.0 0 /
20 0.0 2500.0 0 /
21 0.0 5000.0 0 /
22 0.0 5000.0 0 /
1 0.0 2500.0 0 /
23 0.0 2500.0 0 /
71 2 0.0000000 / NODAL RESTRAINTS - N1 FIXITY DEFLECTION
72 2 0.0000000 /
73 2 0.0000000 /
74 2 0.0000000 /
88 2 0.0000000 /
89 2 0.0000000 /
90 2 0.0000000 /
91 2 0.0000000 /
2 1 0.0000000 /
3 1 0.0000000 /
4 1 0.0000000 /
5 1 0.0000000 /
19 1 0.0000000 /
20 1 0.0000000 /
21 1 0.0000000 /
22 1 0.0000000 /
23 1 0.0000000 /
70 2 0.0000000 /
1 1 0.0000000 /
92 2 0.0000000 /

```

diskpc.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE DSKSPC - SUBROUTINE TO GET THE DISK FREE SPACE AVAILABLE.
PAGE ,132

; (C) Copyright Microstress Corporation 1985, 1986

COMMENT *
DSKSPC is a routine designed to be called from FORTRAN as a subroutine
to determine the available disk free space.

(continued)

```

Mode of use:          call DSKSPC (drive,nbyt)
where
    drive = integer variable containing the drive number (0=default,
              1=A, etc...)
    nbyt = integer-4 variable containing the number of free bytes
            available in the disk.
*
```

```

SUBTTL FORMAL DECLARATIONS
PAGE
```

```

csdsk  SEGMENT 'CODE'
        ASSUME CS:csdsk
```

```

SUBTTL DSKSPC - EXECUTABLE CODE
PAGE
```

```

PUBLIC DSKSPC
DSKSPC PROC FAR
```

```

        PUSH BP
        MOV BP,SP
        PUSH DS
; Get the drive number.
        LDS BX,DWORD PTR [BP+10]
        MOV dx,[bx]
; Call the system function.
        mov ah,36h
        int 21h
; Handle parameters from calling program
        xor dx,dx
        cmp ax,0FFFFh
        je  exit
        mul cx
        mul bx
; Everything done except housekeeping.
exit:
        LDS si,DWORD PTR [BP+6]
        MOV [si],ax
        inc si
        inc si
        MOV [si],dx
        POP DS
        MOV SP,BP
        POP BP
        RET 8h
```

```

DSKSPC ENDP
csdsk  ENDS
```

```

END
```

```

movstr.asm
```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

```

TITLE MOVSTR - SUBROUTINE TO MOVE A STRING INTO ANOTHER STRING
PAGE ,132
```

```

; (C) Copyright Microstress Corporation 1984, 1985, 1986
```

```

COMMENT *
```

```

MOVSTR is a routine designed to be called from FORTRAN as a subroutine
to move a string into another string.
```

```

Mode of use:
```

```

    call MOVSTR (deststr,inidest,eosf,sourstr,inisour,length)
```

```

where
```

```

    deststr = destination string name.
    inidest = character in "deststr" where the new string will go.
    eosf = end-of-string flag to extract (=1) or to insert (=0).
    sourstr = source string name.
```


inlsour = character in "sourstr" where the substring starts.
length = number of characters to be contained by the substring.

This routine does not check the proper bounds of the indices "inidest"
and "inlsour", but it does check the validity of "length".

*

SUBTTL FORMAL DECLARATIONS

PAGE

```
csmovs SEGMENT 'CODE'
ASSUME CS:csmovs
```

SUBTTL MOVSTR - EXECUTABLE CODE

PAGE

```
PUBLIC MOVSTR
MOVSTR PROC FAR
```

```
    PUSH BP
    MOV BP,SP
    PUSH DS
    PUSH ES

; Handle parameters from calling program
    LES BX,DWORD PTR [BP+22]
    MOV cx,ES:[BX]

; Check positive request for location in destination string.
    push cx
    cmp cx,0
    jle outbounds
    LES di,DWORD PTR [BP+26]
    mov bx,di
scanend1:
    mov al,es:[bx]
    cmp al,0
    je outbounds
    cmp al,6
    je outbounds
    inc bx
    loop scanend1

; The location request for the destination string is legitimate.
    pop cx
    dec cx
    add di,cx
    LDS BX,DWORD PTR [BP+10]
    mov cx,ds:[bx]

; Check positive request for location in source string.
    push cx
    cmp cx,0
    jle outbounds
    LDS si,DWORD PTR [BP+14]
    mov bx,si
scanend2:
    mov al,ds:[bx]
    cmp al,0
    je outbounds
    cmp al,6
    je outbounds
    inc bx
    loop scanend2

; The location request for the source string is legitimate.
    pop cx
    dec cx
    add si,cx
    jmp numchars

; There was a non-legal request. Exit leaving the strings untouched.
outbounds:
    pop cx
    jmp exit

; Continue with rest of parameters.
numchars:
    mov ax,ds
    LDS BX,DWORD PTR [BP+6]
    mov cx,ds:[bx]
    cmp cx,0
```

(continued)

```

        jle exit
        LDS BX,DWORD PTR [BP+18]
        mov dx,ds:[bx]
        push dx
        mov ds,ax
; Determine the length of the source string.
        xor dx,dx
        mov bx,si
scanensour:
        mov al,ds:[bx]
        inc dx
        inc bx
        cmp al,0
        jz endsour
        cmp al,6
        jz endsour
        cmp dx,cx
        jl scanensour
        inc dx
endsour:
        dec dx
        mov cx,dx
; Determine the physical length of the destination string.
        xor dx,dx
        mov bx,di
scanendest:
        mov al,es:[bx]
        inc dx
        inc bx
        cmp al,0
        jz enddest
        cmp al,6
        jnz nologend
        pop ax
        mov ax,1
        push ax
nologend:
        cmp dx,cx
        jl scanendest
        inc dx
enddest:
        dec dx
        mov cx,dx
; Move the substring from source into destination.
        rep movsb
; Add an end-of-string byte if extract (=1) but not if insert (=0).
        pop dx
        cmp dx,0
        jz exit
        mov al,6
        mov es:[di],al
; Everything done except housekeeping.
exit:
        POP ES
        POP DS
        MOV SP,BP
        POP BP
        RET 18H

MOVSTR  ENDP
csmovs  ENDS

END

```

memava.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE MEMAVA - FUNCTION TO DETERMINE HOW MUCH MEMORY CAN BE ADDED
PAGE ,132

; (C) Copyright by Microstress Corporation 1985, 1986
; Written by Fernando Garcia-Loygorri


```

COMMENT *
MEMAVA is a routine designed to be called from FORTRAN as a function
to determine how much memory is available to the last common block as
dynamic memory.

Mode of use:
                size = MEMAVA (var)
where
    var = variable name of the last array in the last common block.
    size = name of the 4-byte integer variable receiving the return
           value.
*
```

```

SUBTTL FORMAL DECLARATIONS
PAGE
```

```

csmema SEGMENT 'CODE'
        ASSUME CS:csmema
```

```

SUBTTL MEMAVA - EXECUTABLE CODE
PAGE
```

```

PUBLIC MEMAVA
memava PROC FAR
```

```

        PUSH BP                                ; Save registers.
        MOV BP,SP
        PUSH DS
; Get total amount of room in machine.
        mov ax,40h                            ; Point to the location in memory where
        mov ds,ax                            ; DOS keeps the total amount of memory:
        mov bx,13h                            ; Segment 40h, offset 13h-14h.
        mov ax,[bx]                          ; Get number of K in AX.
        mov cl,6                              ; Multiply by 64 to get the total number
        shl ax,cl                            ; of paragraphs (groups of 16 bits).
; Get address of parameter from program.
        LDS BX,DWORD PTR [BP+6]              ; Address of the parameter.
        mov dx,ds                             ; Store the segment to add later.
        mov cl,4                              ; Divide by 16 to get the number of
        shr bx,cl                            ; paragraphs in the offset.
        inc bx                                ; Increment for the one partially used.
        add dx,bx                            ; Add it to the segment.
        sub ax,dx                            ; Subtract it from the total.
; Convert to words in DX:AX because that is where FORTRAN expects the return of
; a 4-byte integer.
        push ax                               ; Save it for later.
        mov cl,13                            ; Set to isolate top 3 bits.
        shr ax,cl                            ; Isolate them by shifting.
        mov dx,ax                            ; Move them to DX.
        pop ax                               ; Recover it.
        mov cl,3                             ; Multiply by 8 (words/paragraph) by
        shl ax,cl                            ; shifting 3 places.
; Everything done except housekeeping.
        POP DS                                ; Recover the registers saved.
        MOV SP,BP
        POP BP
        RET 4                                ; Discard the input parameter.
```

```

memava ENDP
csmema ENDS
```

```

END
```

```

chopwr.asm
```

```

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.
```

```

TITLE CHOPWR - SUBROUTINE TO CHECK IF A FILE IS READY TO BE OPEN
PAGE ,132
```

```

; (C) Copyright Microstress Corporation 1984, 1985, 1986
```

(continued)

```

COMMENT *
      Mode of use:
              call CHOPWR (name,flag)
      where
              name = string with the name of the file to be checked.
              flag = integer to show success (=0) or error (<>0).
      *

SUBTTL  FORMAL DECLARATIONS
PAGE

cschow  SEGMENT 'CODE'
        ASSUME  CS:cschow

SUBTTL  CHOPWR - EXECUTABLE CODE
PAGE

PUBLIC  CHOPWR
CHOPWR  PROC  FAR

        PUSH    BP
        MOV     BP,SP
        push    ds
        LDS     DX,DWORD PTR [BP+10]
        MOV     cx,20h
        mov     ah,3Ch
        int     21h
        jb      exit
        mov     bx,ax
        mov     ah,3Eh
        int     21h
        xor     ax,ax

exit:
        LDS     BX,DWORD PTR [BP+6]
        mov     [bx],ax
        pop     ds
        MOV     SP,BP
        POP     BP
        RET     8

CHOPWR  ENDP
cschow  ENDS

END

```

brochure.doc

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

===INTRODUCTION=====

The following manual is presented in a very basic format because it was designed for printing on any printer. This procedure was followed to allow potential users of the MICROSAFE 2-D package to print the on-disk brochure as an assist in evaluating the programs.

The information contained in this brochure has been extracted from the 139 page, 8.5 x 11 inch manual supplied with the purchased programs. The manual is professionally typeset in a custom binder and contains much more information about the use of the programs that is summarized here.

The programs are written for in-core finite element solutions of models with any combination of up to 400 nodes, 500 plates, 600 beams and 60 fasteners. The beam elements may have cross-sectional area and moment of inertia and the plate elements may be triangular or quadrilateral membranes.

This software package was written for the IBM-PC running DOS 2.0. It may also be used with the rest of IBM computers of the PC line, including the PC/AT. Other computers will be able to run MICROSAFE 2-D as long as they are closely compatible with the IBM-PC.

The minimum IBM-PC configuration required is one double-sided disk drive and the availability of 448K memory. Bigger memory configurations (up to 704K or the maximum recognized by DOS, whichever is lower) are fully exploited by MICROSAFE 2-D to allow the analysis of models with larger bandwidths. In addition, the plotting program will require a combination graphics monitor/IBM Graphics Display Adapter.

For efficiency reasons MICROSAFE 2-D is a two-part package:

Part one is a program called SAFEPLLOT used to present graphical displays of the model along with all input data for accuracy verification and assistance in correcting any data errors.

The SAFEPLLOT program generates plots of the nodal diagrams and structural element diagrams as well as the properties, applied loads, imposed deflections, restraints and coordinate values for all elements of the model. Windows are available for the user to see enlarged views of parts of the model. Details too small to be resolved in a larger scale view may thus be displayed.

Part two is a set of two programs, called SAFESOLV and SAFESOLB, used to conduct the actual structural analysis using the stiffness method finite element analysis for 2-dimensional structures. SAFESOLB is a special version of SAFESOLV used to solve a unique class of problems.

The analysis programs SAFESOLV and SAFESOLB present a commented listing of the input describing the model (optional) and tables of internal loads and stresses for each element. These tables include corner forces tabulated for each element as well as for all nodes, the latter showing the overall equilibrium of forces for the model. Deflections are also presented for all nodes.

In addition, the package includes example input files, and other miscellaneous files to set up the computer and printer and to generate a MICROSAFE 2-D input file.

The package has been written in the FORTRAN language and makes extensive use of machine language routines to increase the speed and the capabilities of the programs.

===TABLE OF CONTENTS OF THE MICROSAFE 2-D MANUAL=====

This is the table of contents of the MICROSAFE 2-D User's Manual. Most of the information in this brochure has been extracted and condensed from the manual.

TABLE OF CONTENTS	iii
PREFACE	1
1 INTRODUCTION	3
1.1 Organization of the MICROSAFE 2-D disks	4
1.2 Set-up instructions for the production disks	6
1.2.1 Setting-up the SAFEPLLOT disk	6
1.2.2 Setting-up the SAFESOLV disk	7
1.2.3 Setting-up the SAFESOLB disk	8
1.2.4 Setting-up a data disk	8
1.3 Usage and copying of the MICROSAFE 2-D package	8
2 DESCRIPTION OF FINITE ELEMENTS	11
2.1 Nodes	11
2.2 Beams	12
2.3 Plates	14
2.4 Fasteners	15
3 INTRODUCTION TO MODELING	17
4 DESCRIPTION OF THE INPUT DATA FILE	19
4.1 Overview of the input file	19
4.2 Input file summary	21
4.3 Number of nodes	22
4.4 Number of materials	22
4.5 Number of beams	22
4.6 Number of plates	23
4.7 Number of fasteners	23
4.8 Number of loaded nodes	23
4.9 Number of specified displacements	24
4.10 Node definition	25

(continued)

4.11 Material definition	26
4.12 Beam definition	27
4.13 Plate definition	29
4.14 Fastener definition	30
4.15 Node loads definition	31
4.16 Freedom restraint definition	32
5 CREATING THE MODEL - INPUT GENERATION	33
6 PLOTTING THE MODEL - SAFEPLLOT PROGRAM	35
6.1 Starting the SAFEPLLOT program	35
6.2 Running the SAFEPLLOT program	36
6.3 SAFEPLLOT program command summary	38
6.4 SAFEPLLOT program command description	39
6.4.1 Plot a new file	39
6.4.2 Plot beam elements	40
6.4.3 Display beam area values	40
6.4.4 Display beam moment of inertia values	41
6.4.5 Display beam material codes	41
6.4.6 Display beam numbers	42
6.4.7 Clear screen	42
6.4.8 Plot beam distributed loads	43
6.4.9 Display distributed beam load values	43
6.4.10 Enlarge window	44
6.4.11 Display fasteners	45
6.4.12 Display fastener area values	45
6.4.13 Display fastener numbers	45
6.4.14 Display fastener stiffness values	46
6.4.15 Plot node loads	46
6.4.16 Display node load values	47
6.4.17 Move window	48
6.4.18 Plot nodes	48
6.4.19 Display node coordinates	49
6.4.20 Display node numbers	49
6.4.21 Plot plates	49
6.4.22 Display plate material codes	50
6.4.23 Display plate numbers	50
6.4.24 Print display image	50
6.4.25 Display plate thickness values	50
6.4.26 Quit program	51
6.4.27 Plot restraints	51
6.4.28 Display restraint values(imposed displacements)	51
6.4.29 Shrink window	52
6.4.30 Display the model World	52
7 ANALYZING THE MODEL	53
7.1 Choosing between the SAFESOLV and SAFESOLB programs	53
7.2 Starting the solution program	54
7.3 Running the solution program	56
8 DESCRIPTION OF THE OUTPUT FILE	59
8.1 Header	59
8.2 Listing of input data	60
8.3 Listing of output data	62
8.4 Node displacements	63
8.5 Beam corner forces	64
8.6 Beam loads and stresses	65
8.7 Plate corner forces	66
8.8 Plate load intensities and stresses	67
8.9 Plate stresses at node points	68
8.10 Fastener forces and stresses	68
8.11 Node internal forces and reactions	69
9 PROGRAM MESSAGES	71
9.1 Warning messages	71
9.2 Error messages	77
9.2.1 Recoverable errors	77
9.2.2 Non-recoverable error messages	79
A REFERENCES	87
B MICROS SAFE 2-D SUMMARIES	89
B.1 SAFESOLV program	89
B.2 SAFEPLLOT program	90
B.3 Input file format	91
C EXAMPLES	93
C.1 Beam With Multiple Supports	94
C.2 Skin Lap-splice With Fasteners	98
C.3 Spar-bulkhead Intersection	106
C.4 Built-up Transport Airplane Frame	111
C.5 Lane Problem	121
D PURCHASE FORM	131
INDEX	133

===Usage and copying of the MICROSAFE 2-D package=====

The BYTE copy of the MICROSAFE 2-D source code is for private use and is not to be distributed in compiled form. Please use the complete MICROSAFE 2-D discs for distribution since they are safer and more complete for a new user.

Users of MICROSAFE 2-D should be aware that both the computer programs and the User's Manual that make up the MICROSAFE 2-D package fall under the scope of the 1976 Copyright Act and that MICROSTRESS Corporation holds a copyright on them.

MICROSTRESS Corporation does not sell buyers a license to use its programs but actually sells copies of them in a manner similar to the way publishing companies sell books.

The MICROSAFE 2-D programs are not copy-protected. This allows the legal owner to make a back-up copy as a protection against accidental destruction.

The original disk, the back-up copy or a hard disk copy may be used, BUT ONLY ONE OF THE ABOVE MAY BE USED AT THE SAME TIME.

Any other use would be a violation of the copyright because the user paid for only one copy. On the other hand, there is no restriction on who uses the package or which computer runs it, but if the user wants to run simultaneously several copies of the program each copy must be purchased. Contact MICROSTRESS Corporation about conditions for multiple-copy purchases.

The MICROSTRESS Corporation also authorizes the distribution of copies of the disks included in the MICROSAFE 2-D package, to prospective buyers, only if the following stipulations are satisfied:

- * The distribution of copies is for evaluation purposes ONLY. Use of the programs on a regular basis and/or with the intention of applying results is only permitted to those purchasing the MICROSAFE 2-D package.
- * No charge whatsoever may be collected, in any form, for the distribution.
- * The recipient of the evaluation copy MUST be instructed by the donor, in advance, of these conditions.
- * The copy must be COMPLETE, containing ALL the files included in the manufacturer's release. This is to prevent the evaluation of incomplete copies.
- * THE PRINTED MANUAL IS NOT TO BE REPRODUCED. The on-disk documentation is adequate for evaluation with the many examples included. The printed manual contains additional documentation that is necessary for reliable applications of the MICROSAFE 2-D software to the design of structures.

Purchasers of this software will be registered and notified of any enhancements and/or corrections that may be developed as long as their mailing addresses are kept up to date.

The MICROSTRESS Corporation welcomes comments of any kind about the MICROSAFE 2-D programs and this user's manual. We encourage the users of this package to write to us relating experiences and suggestions to make it a better product.

As a follow up, a three dimensional version of this program is currently in development.

LIMITATIONS OF LIABILITY FOR USE AND THE RESULTS OF USE

The MICROSAFE 2-D software package has been tested by numerous engineers for problems of the type shown in the examples in Appendix C of the printed manual. New applications may uncover application problems beyond these and many other tests that have been run. If problems are detected, please notify the MICROSTRESS Corporation and an attempt will be made to correct the problem.

(continued)

These programs are provided on an 'as is' basis and the MICROSTRESS Corporation does not guarantee, warrant or make any other representation regarding the use of these programs or the use of results generated from these programs. The user is responsible for the engineering validation of the program's mathematical results and the suitability of this analysis to the problem being analyzed.

===DESCRIPTION OF FINITE ELEMENTS=====

===Nodes===

Nodes are used to define the shape of the finite element model. A node is dimensionless and has the properties of physical coordinates only.

Each node is identified by a unique number, an integer between one and the total number of nodes, both inclusive.

Concentrated loads and node restraints are applied to the nodes of the model to define the load conditions to be analyzed. An adequate number of restraints must be defined to prevent the complete model from being able to translate in the x-direction, the y-direction and from any inplane rotation.

The idealized model deforms under the effect of the applied loads and restraints. The deformations are represented by translations and rotations of the nodes in the global coordinate system.

The node restraints generate reactions from the combination of applied loads and imposed deflections. These reactions are presented in the global coordinate system, along with any residual loads at nodes that are not restrained, in the SAFESOLV model solution printed output.

MICROSAFE 2-D allows the user to create a model with up to 400 nodes. The amount of memory (RAM) above 448K increases the available bandwidth for the model rather than allowing more nodes.

Not every node needs to be connected to the structural elements of the model since the program will ignore nodes that are disconnected from all elements.

===Beams===

Beam elements are used to define axial and bending stiffnesses between any two nodes. The shear stiffness terms are omitted since they are insignificant for the vast majority of problems. Because this version of the MICROSAFE 2-D software is used to analyze models that are two-dimensional the only stiffnesses that are defined are contained within the plane of the model.

The sequence of the nodes defining the beam determines the local coordinate system. The properties of the beam and its internal loads are defined in the context of the local coordinate system.

The cross-sectional area of the beam may be defined as zero to disconnect the element from the rest of the model. The moment of inertia may also be defined as zero to reduce it to a rod.

Beam loads are distributed loads and are defined by values of force per unit of length at each end of the beam. These distributed load end values may be of different magnitude and sign at each end to represent any desired loading. A linear variation is assumed between the two ends of the beam. The loads are converted to statically equivalent node loads at each node of the beam except for the case of rods, where the moments are neglected due to the lack of moment of inertia.

Up to 600 beams may be included in a model to be analyzed with MICROSAFE 2-D.

===Plates===

Plates are used to define the membrane and shear stiffness connecting three or four nodes. The user is free to use any sequence to define the nodes of a plate.

The geometric properties of the plate are completely defined by the node locations and the thickness, which is assumed to be constant throughout the plate.

The plates idealized with MICROSAFE 2-D are isotropic, so the material properties in the two orthogonal directions are equal.

The triangular plates used in conventional finite element analysis systems are simple but do not provide precisely symmetrical properties. To avoid this, MICROSAFE 2-D allows the use of quadrilateral plates. The plate is divided into four triangular plates connecting the centroid with each of the four sides of the original quadrilateral plate. The stiffness contribution of each of the four triangular plates is computed as usual and included in the global stiffness matrix.

A beneficial side effect is a dramatic increase in the effective resolution of the grid (by as much as 40% in some models). The results demonstrate that the quadrilateral plates should be used whenever possible in finite element solutions for more accurate results.

Plates may overlap other plates, as when fasteners are used. Since this is a two-dimensional program, the eccentricity of the plates in the thickness direction will be neglected.

The plate internal axial loads and stresses are defined in the x-axis direction and in the y-axis direction and shears are defined at 45 degrees to the x and y axis. Corner forces are also defined for each node of the plate and are relative to the global coordinate system.

MICROSAFE 2-D allows a model to contain up to 500 plates.

===Fasteners===

Fasteners are used to define a connection between any two nodes. The fastener element transfers an inplane load between the two nodes based on the relative deflection between these nodes and the stiffness of the fastener element. The stiffness is defined for the fastener element explicitly by input and is therefore not a function of the distance between the nodes.

The fastener area is only used to allow fastener shear stresses to be calculated and is irrelevant for the stiffness calculation.

Up to 60 fasteners may be included in a model to be analysed through MICROSAFE 2-D.

===DESCRIPTION OF THE INPUT DATA FILE=====

===Overview of the input file===

The three programs that make up the MICROSAFE 2-D software package use a common input data file containing all the relevant information about the finite element model to be analyzed.

When the plotting program is used to check out the data in a given data file, the user knows it will be correct when run with SAFESOLV.

The input data file is a group of lines of text in a given order that may be created with any editor or wordprocessing system that produces an ASCII output file.

The MICROSAFE 2-D programs scan the input file sequentially ignoring all lines that do not contain a slash character (/); this way the user can mix lines with comments throughout the entire file for future reference.

The data area in the lines of the input file is filled with individual numerical entries. Each entry must be separated from the others by one or more blank spaces and no other punctuation symbols (commas, dashes, ...) are allowed.

For each given line of the input file the programs expect a certain number of numerical entries and each entry is supposed to be of a certain type and fall within a certain range of values. The entries

(continued)

not complying with these rules will be rejected and a detailed error message will be displayed.

===Input file summary===

The input lines must be ordered in the following manner:

Type	Format	Description
Model size definition	NN /	Number-of-nodes
.	NM /	Number-of-materials
.	NB /	Number-of-beams
.	NP /	Number-of-plates
.	NF /	Number-of-fasteners
.	NL /	Number-of-loaded-nodes
.	NR /	Number-of-imposed-restraints
Node definition (One line per node until all nodes are defined)	NI NX NY /	Node X-coordinate Y-coordinate
Material definition (One line per material until all materials are defined)	MI ME MP /	Material Young's-modulus Poisson-ratio
Beam definition (One line per beam until all beams are defined)	BI BN1 BN2 BA BMI BM BQ1 BQ2 /	Beam Node-1 Node-2 Area Moment-of-inertia Material D.Load-at-node-1 D.Load-at-node-2
Plate definition (One line per plate until all plates are defined)	PI PN1 PN2 PN3 PN4 PT PM /	Plate Node-1 Node-2 Node-3 Node-4 Thickness Material
Fastener definition (One line per fastener until all fasteners are defined)	FI FN1 FN2 FA FS /	Fastener Node-1 Node-2 Area Stiffness
Node loads definition (One line per loaded node until all node loads are defined)	LNI PX PY MZ /	Node X-load Y-load Z-moment
Freedom restraint definition (One line per restrained freedom until all restraints are defined)	RNI RC FD /	Node Freedom Displacement

All the lines not containing a slash character (/) will be treated as comment lines and ignored.

Each input line is described in detail in the manual. The following is a typical example:

===Node definition===

Input: NI NX NY /

Description:

NI is the number of the node defined by the subsequent coordinates.

Format: Integer or whole number

Range: $1 \leq NI \leq NN$

Units: None

Remarks: No two nodes are allowed to have the same number and no gaps are allowed in the node number sequence which starts with one. See note below.

NX is the x-coordinate of the node along the x-axis.

NY is the y-coordinate of the node along the y-axis.

Format: Real

Range: $-10E19 \leq NX \text{ or } NY \leq 10E19$

Units: Length

Remarks: Several nodes can have the same coordinate values, as when overlapping structure is defined.

Repeat this input line until all nodes are defined.

===PLOTting THE MODEL - SAFEPLoT PROGRAM=====

===Starting the SAFEPLoT program===

The input data is verified with graphical displays of the model elements and with properties superimposed as defined by interactive user input.

The following command will run the SAFEPLoT program:

```
A>d:SAFEPLoT m c i=inpfspec
```

The m, c and i=inpfspec parameters are optional and they may be specified in any order.

The m parameter is used to set the monochrome mode. This option is useful when using a combination monitor/display card that does not work well with colors other than black and white.

The c parameter centers all plot values relative to the element centroid rather than make them stack around it. This is useful in fine grid models where the printed value is large relative to the element.

The i=inpfspec option may be used to specify the file to be plotted. The inpfspec may be any valid DOS file name with drive specification, directory path, name and type as desired. Only the name is required. If not specified, the program will revert to the default drive and to the current directory and an extension of .INP will be automatically added by SAFEPLoT.

For example:

```
A>B:SAFEPLoT m i=beam C
```

The program is in drive B: and the input file is BEAM.INP in the default drive A:. Both the monochrome and the centering options have been selected.

===SAFEPLoT program command summary=====

Command: CODE options

CODE specifies the action to be taken.

CODE	Description of function selected with CODE
A	Run Another data file --- program prompts for file name.
B	Plot Beam elements as magenta lines.
BA	Display Beam Area values in magenta.
BI	Display Beam moment of Inertia values in magenta.
BM	Display Beam Material codes in magenta.
BN	Display Beam Numbers in magenta.
C	Clear screen of all plotted and displayed data.
D	Plot Distributed beam loads as a magenta surface.
DV	Display Distributed beam load Values in magenta.
E	Get in the Enlarge mode.
	B : Move bottom of the box up.
	L : Move left side of the box to the right.
	R : Move right side of the box to the left.
	T : Move top of the box down.
	- : Reverse direction of movement.
F	Plot Fasteners as cyan lines.
FA	Display Fastener Area values in cyan.
FN	Display Fastener Numbers in cyan.
FS	Display Fastener Stiffness values in cyan.
L	Plot nodal Load vectors as cyan arrows.
LV	Display nodal Load Values in cyan.
M	Move the window --- The cursor keys may also be used.
N	Plot Nodes as cyan circular symbols.
NC	Display Node Coordinate values in cyan.
NN	Display Node Numbers in cyan.
P	Plot Plates as a white surface.
PM	Display Plate Material codes in white.
PN	Display Plate Numbers in white.
PR	Print display without the "Option to plot?" prompt.

(continued)

PT Display Plate Thickness values in white.
 Q Quit --- used to terminate this program run.
 R Plot type of nodal freedom Restraints as white symbols.
 RV Display node Restraint Values in white.
 S Shrink display grid to previous window.
 W Display the World --- the complete model.

Options are parameters used to specify the request with more detail when necessary. There are two types of options:

Range: Used by commands that plot elements or properties of elements, a range is a compact form of specifying a list of elements for which the request is made. The range is specified with two integers, the first and last values of the range. If both entries are missing, it will plot the entire range, while if only one entry is present, that item will be plotted.

Direction: Required by the Move command to specify the direction of movement from the present window. It is specified by a single digit.

Here are some examples:

Option to plot? BA => Display all the beam area values.

Option to plot? nn 1 18 => Display the node numbers for all the nodes in the range from 1 to 18, both inclusive.

Option to plot? F 23 => Plot the fastener numbered 23.

Each command is described in detail in the manual, in a similar way to the following examples:

Move window

Command: M CODE

Range: CODE values are 1, 2, 3, 4, 6, 7, 8, or 9.

Effect: Moves the window to the adjacent window location according to the code value. The code values indicate the direction of movement in the numeric keypad from the number 5 key to the key with the number specified by CODE.

Remarks: The display window may be moved in one of the eight directions around it by using the key layout in the numeric pad to specify the direction:

7 = up and left	8 = up	9 = up and right
4 = left	5	6 = right
1 = down and left	2 = down	3 = down and right

There will be an overlap of about 10% between adjacent windows to allow a good indexing of the images both visually and to create a mosaic of images obtained in a printer (see the PR command). If the desired movement of the window amounts to half-frame or less, the Enlarge command should be used (see 6.4.10).

This command does not enlarge the image and so the Shrink command will not return the window back to the previous frame from where it was moved but to the previous frame from where the image was actually enlarged.

If numbers are to be entered from the numeric keypad, the Num Lock key may need to be toggled to switch the numeric key pad from the cursor control state to the numeric entry state.

SPECIAL NOTE: The cursor keys uparrow, leftarrow, rightrightarrow and downarrow and the Home, Pg Up, Pg Dn and End keys will also move the window in the same manner as the Move command described above. They are less cumbersome to use, requiring only to press the key in desired direction relative to the central key with the number 5.

===Plot nodes===

Command: N init last

Range: 1 <= init, last <= NN

Effect: All the nodes in the specified range of nodes will be plotted as rings in cyan color centered on the location of the node.

If last is omitted only the node init will be displayed.

If both init and last are omitted all nodes will be plotted.

===ANALYZING THE MODEL=====

Starting the solution program

The following command will run the analysis program

A>d:SAFESOLV i=inpfspec o=outfspec e s

The e and s parameters may be in any order including being mixed with the input and output options. The e is used to print an echo of the input data in the output file and s is used to list the output file on the screen at the same time that it is created.

The i=inpfspec and o=outfspec parameters are also optional inputs. The i=inpfspec option is used to set the input file name. The o=outfspec option is used to specify the file to be used for all printed output. The sequence of the i=inpfspec and o=outfspec options may be reversed. If one or both of these parameter keys and associated filenames are omitted the program will ask for the missing file names during the run.

The inpfspec and outfspec may be any valid DOS file name with optional drive specification, directory path and type as desired. If a file type is not specified, the program will use .INP for the input file type and .OUT for the output file type.

The output file may be left completely unspecified (as long as O= is included) and the program will automatically use the same drive, directory path and name as the input file and the .OUT extension.

As a special case the output file data can be routed to any legal output device known to DOS, in which case no output diskfile will be generated. An example of this is the screen device CON: or the line printer LPT1:. This is useful when screen output or printed output, but not diskfile, is desired or where insufficient hard disk or RAM disk space is available. The program will ask for another disk if the output fills up the existing floppy disk (see below) but this option is not available with a fixed disk or with a RAM disk.

For example:

B>B:SAFESOLV i=a:beam o= e => The program is in drive B:
(also the default), the input file is BEAM.INP in drive A: and the
output file is BEAM.OUT in the same drive A:. The echo option has been
selected.

C>SAFESOLV i=Beam o=A: => The program is in the
default drive C:, the input file is BEAM.INP in the default drive C:
and the output file is BEAM.OUT in drive A:. None of the echo or
screen options have been selected.

Running the solution program

After the logo of the program is displayed in the screen the program will evaluate the parameters passed in the command line ---if any--- and it will prompt for the input and output file names if missing.

After the input file has been specified, SAFESOLV will try to locate it. If the program cannot find it, it will print the following message on the screen:

(continued)

ERROR : File "inpfspec" cannot be found. Try again.

The program will then prompt for the proper file specification again.

After the output file has been specified, SAFESOLV will try to set it up.

If the program cannot open the output file, it will print the following message on the screen:

ERROR : File "outfspec" cannot be open. Try again.

The program will then prompt for the proper file specification again.

Once SAFESOLV has located the input file, it will proceed to read its contents and will verify its validity. The screen will show the progress as the data is read from the input file with a string of words progressing across the screen as follows:

Size...Nodes...Materials...Beams...Plates...Fasteners...Loads...
Restraints...End

Each keyword corresponds to a block of lines in the input file. The word End marks the completion of the process of reading the input file.

After reading the input file, it will start the solution of the resulting stiffness matrix.

The structural relationships in the model are represented mathematically by a system of simultaneous equations. The number of equations in the system is equal to the number of degrees of freedom and is reported by the SAFESOLV program prior to solving the system. In plane stress models like those analyzed with the MICROSAFE 2-D package there are three degrees of freedom per node.

The SAFESOLV programs keep the user informed of the progress in each step by a plotted bar progressing between end marks on the screen. This helps to give an idea of the amount of time required to finish the job.

The entire process will take only a few seconds for simple models like the beam shown in Appendix C or several minutes for bigger models. For example, it takes approximately 5 minutes for 100 node, well-ordered models like the spar-bulkhead or frame examples in the same Appendix.

The 8087 coprocessor chip significantly reduces the above solution times. If installed, all the programs in the MICROSAFE 2-D package will take advantage of its presence. The switch to the 8087 is automatically performed by the programs and no user intervention is required.

The solution time for a matrix is very sensitive to the bandwidth encompassing the element stiffness terms in the stiffness matrix. The bandwidth depends on the order selected for the node numbering.

Once SAFESOLV has solved the system of simultaneous equations, it will proceed to write the results to the output file. The screen will show the progress as the data is written with a progression of words like:

Displacements...Beams...Plates...Fasteners...Reactions...End

The program then finishes by reporting the total amount of time required to analyze the model for future reference and returns control to the operating system.

===DESCRIPTION OF THE OUTPUT FILE=====

The SAFESOLV programs report all the information in a single output file. The name and destination ---drive, directory path--- of the output file is specified by the user at the beginning of each run.

The output file is a plain text file, in ASCII form, and does not contain any special control characters other than the standard carriage-return/line-feed at the end of each line.

The first lines of the output file generated by SAFESOLV contain a header that is handy to quickly identify the run. For example:

```
M I C R O S A F E --- STRUCTURAL ANALYSIS BY FINITE ELEMENTS
Version: SAFESOLV (2-D) Rel. 1.0   4/02/1985  1:00:00
```

```
Input data file : A:BEAMTEST.INP
Output data file : C:BEAMTEST.OUT
```

It shows the date and time of day when the file was created and the data files involved. It also displays the version and release number of the MICROSAFE 2-D package used.

The input data is displayed in labeled tables as the program progressively reads the input file. All short form inputs, input positional values and restraint codes are enhanced/translated to form a complete readable model data description.

===Listing of input data===

The program starts by displaying the parameters that define the size of the model:

SIZE OF THE STRUCTURE

```
Number of nodes           : 25
Number of materials       : 2
Number of beams           : 24
Number of plates          : 36
Number of fasteners       : 0
Number of loaded nodes    : 7
Number of restrained degrees of freedom : 3
```

and continues with the node coordinates definition:

NODE COORDINATES

Node	Coordinate X	Coordinate Y
1	.00000	.00000
2	20.00000	6.00000
3	40.00000	12.00000
4	60.00000	18.00000
5	10.00000	7.50000

In the same order that they are specified in the file, not necessarily in a sorted form. The following table presents the material codes definition:

MATERIAL PROPERTIES

Code	Young's modulus	Poisson's ratio
1	10500000.	.33000
2	10500000.	.33000
3	10700000.	.30000
4	400000.	.11000

and then the program displays the tables with the properties for the different types of elements--- beams, plates and fasteners--- in this order. If one or more types of elements are missing from the model definition, the corresponding tables will not be included.

The table containing the beam properties looks like:

BEAM DATA

Beam	I	J	Length	Area	M.Inertia	Material	Distributed Loads
1	1	2	20.881	.2000	.00020	1	.000 .000
2	2	3	20.881	.2000	.00000	1	.000 .000
3	3	4	20.881	.2000	.00512	2	.000 .000
4	8	9	13.396	.2200	.00000	2	.000 2500.000
5	9	10	13.396	.2200	.06250	1	2500.000 5000.000

(continued)

6	10	11	13.396	.2200	.12500	1	5000.000	5000.000
7	15	16	6.100	.2400	.00000	1	.000	.000

The labels I and J are used to represent the end nodes of the beam. After the beam data table comes the table with the definition of the plates:

PLATE DATA

Plate	I	J	K	L	Thickness	Material
1	1	2	5	0	.05000	3
2	1	5	8	12	.05000	3
3	2	3	6	0	.05500	4
4	2	9	5	0	.06000	4

The labels I, J, K and L are used to represent the corner nodes of the plate.

The last element table will contain the information about the defined fasteners:

FASTENER DATA

Fastener	I	J	Area	Stiffness
1	7	22	.250000	5000000.
2	8	23	.250000	0.

WARNING : The fastener 2 has been disconnected from the model.

3	9	24	.250000	5000000.
---	---	----	---------	----------

As is the case with the beams and plates, a warning message will be included when a given element is disconnected from the model.

The labels I and J are used to represent the end nodes of the fastener.

The last tables of the echoed input data correspond to the node loads and the node restraints and they look like this:

NODE LOADS

Node	PX	PY	MZ
4	.00	1000.00	.00
11	.00	2000.00	670000.00
18	.00	5000.00	.00
25	5000.00	5000.00	.00

MOVEMENT RESTRAINTS

Node	Type of restraint	Displacement
1	Translation along X axis	.00000
1	Rotation about Z axis	.00000
4	Translation along X axis	.00000
4	Translation along Y axis	.02500
4	Rotation about Z axis	.00000

The labels PX, PY and MZ are used to represent the node loads in the x-axis and y-axis directions and the torque around the z-axis respectively.

===Listing of output data=====

The information generated by SAFESOLV about the state of the deformed model is always written to the specified output file with no options to be specified. Again, to see the output data on the screen the user must specify the s option in the command line.

Like the input data, the output data is displayed in labeled tables as the SAFESOLV program progressively generates it. The following Sections explain in some detail what the different columns in the different tables report.

Because of the huge amount of output data generated by SAFESOLV for even mid-sized models, the program constantly monitors the amount of space available in the disk where the output file is being written to prevent a fatal "disk full" system error which would terminate the run.

If the disk becomes full in the process of writing the output data, SAFESOLV will trigger a recovery process to allow the data to be split across different diskettes. This process will be repeated as many times as necessary.

===Node displacements===

The first table of the output data is always the listing of the displacements of the nodes in the three degrees of freedom. The table has the following format:

NODE DISPLACEMENTS

Node	U	V	Omega
1	.000000	.000000	.000000
2	2.190833	-.000137	-.028178
3	2.190039	-.000230	-.012243
5	.000000	.000000	.000000

Note that the nodes that happen to be disconnected from the model are not included in this table. Node displacements at the indicated node are presented in a right handed Cartesian coordinate system.

The U parameter represents the nodal deflection in the x-direction with positive x being to the right.

The V parameter represents the nodal deflection in the y-direction with positive y being up.

The Omega parameter represents the nodal rotation with positive being defined as counter-clockwise.

===Beam corner forces===

Beam corner forces are presented as loads applied to the indicated ends of the beam elements.

BEAM CORNER FORCES

Beam	I	J	FX1	FY1	MZ1	FX2	FY2	MZ2
1	1	2	-748.	134.	27455.	-212.	-134.	-1736.
3	2	3	212.	134.	1736.	-212.	226.	-21576.
4	3	4	512.	-226.	21576.	-512.	226.	27585.

Note that the beams that happen to be disconnected from the model are not included in this table.

Data for node I, which corresponds to the first node of the beam, correspond to forces and moments with labels ending in the numeral 1. Data for node J, which corresponds to the second node of the beam, correspond to forces and moments with labels ending in the numeral 2.

These forces are defined in the global, right handed, Cartesian coordinate system.

The FXi parameters represent the corner forces in the x-direction with positive x being to the right.

The FYi parameters represent the corner forces in the y-direction with positive y being up.

The MZi parameters represent the end moments around the z-axis with a positive value being counterclockwise.

(continued)

For each beam, the reported corner forces must be in static equilibrium with the applied beam distributed loads. For example, in the case of beam 1 above there is equilibrium in the y-direction with no distributed load applied in that direction, while the distributed load applied in the x-direction amounts to 960 units, balancing FX1 and FX2.

===Beam loads and stresses===

Beam internal loads and stresses are presented in relation to the beam element local coordinate system in the following manner:

BEAM LOADS AND STRESSES

Beam	I	J	PAX	SAX	SH1	SH2	BM1	BM2
1	1	2	-134.	-43.	-748.	212.	-27455.	-1736.
3	2	3	-212.	-55.	-134.	226.	-1736.	-21576.
4	3	4	-226.	-72.	-512.	-512.	-21576.	27585.

Axial load (PAX) represents the force along the axis of the element, which coincides with the local x-axis and is defined as being positive from node I to node J. Axial tension is positive and axial compression is therefore shown as negative.

The axial stress (SAX) follows the same sign conventions as PAX.

The SH1 parameters represent the shear forces at both ends of the beam. The shear at the second node (SH2) is defined along the local positive y-axis, located 90 degrees counter-clockwise from the x-axis. The shear at the first node (SH1) is defined in the opposite direction to SH2.

The BM1 parameters represent the bending moments at the ends of the beam. Positive bending is defined as producing compression in the positive y-surface of the beam. Therefore, the positive bending moment at the second node (BM2) is counterclockwise and

the positive bending moment at the first node is clockwise.

===Plate corner forces===

Plate corner forces are presented as forces applied to the nodes at the corners of the plate element.

PLATE CORNER FORCES

Plate	I	J	K	L	FX1	FY1	FX2	FY2	FX3	FY3	FX4	FY4
1	1	2	5	0	-5751.	-1883.	824.	2306.	4927.	-423.	0.	0.
2	2	3	10	9	-6735.	-2116.	3141.	-125.	2461.	2997.	1132.	-756.
4	2	9	5	0	3292.	-1345.	1310.	1703.	-4602.	-358.	0.	0.
5	2	6	9	0	-3891.	-1370.	2731.	2703.	1160.	-1333.	0.	0.

The plates that happen to be disconnected from the model are not included in this table. Data for node I, which corresponds to the first node defined for the plate, is presented under labels ending in the numeral 1. Data for nodes J, K and L is presented in a similar way.

These forces are defined in the global, right handed, Cartesian coordinate system.

The FXi parameters represent the corner forces in the x-direction with positive x being to the right.

The FYi parameters represent the corner forces in the y-direction with positive y being up.

For each plate, the reported corner forces must be in static equilibrium. The user may check that it is true by adding all the corner forces of each plate in a given direction, as is the case in the example table shown above.

===Plate load intensities and stresses===

The table following the plate corner forces displays the plate load intensities and stresses and looks like:

PLATE LOAD INTENSITIES AND STRESSES

Plate TMAX	I	J	K	L	PIX	PIY	TXY	SX	SY	TAU	SMAX	SMIN
	Angle											
1	2	5	0		1461.	237.	931.	29219.	4740.	18620.	39262.	-5303.
22282.				62.								
2	3	10	9		811.	212.	388.	16225.	4233.	7758.	20035.	424.
9806.				64.								
4	2	9	5	0	1151.	327.	89.	23012.	6542.	1790.	23204.	6349.
8427.				84.								
5	2	6	9	0	683.	173.	676.	13657.	3455.	13515.	23002.	-5890.
14446.				55.								

The PIX, PIY and TXY values represent the plate load intensities in the x, y and xy directions respectively.

The SX, SY and TAU values represent the plate stresses in the x, y and xy directions respectively.

The SMAX, SMIN and TMAX values represent the plate principal stresses: maximum axial stress, minimum axial stress and maximum shear stress.

The Angle parameter shows the orientation of the principal stresses in the global coordinate system.

The principal stress data are calculated with the traditional Mohr's circle method.

The xy direction is used to indicate the 45 degree diagonal between the x and y axes for shear flows (TXY) and shear stresses (TAU).

For quadrilateral plates, the stress matrix is formed by assembling the stress matrices of four individual triangular plate elements. This method prevents the unsymmetrical error associated with the usual stiffness matrix for a two-plate idealization of a quadrilateral plate. The data values presented are an average of the values of the triangular subelements.

===Plate stresses at node points===

The SAFESOLV program presents average stresses for the plate elements at each node. This is done in a table similar to the following example:

PLATE STRESSES AT NODE POINTS

Node	Coord- X	Coord- Y	SX	SY	TAU	SMAX	SMIN	TMAX	Angle
1	.00000	.00000	24478.	8639.	21085.	39081.	-5965.	22523.	55.
2	40.00000	12.00000	22816.	4372.	11074.	28005.	-817.	14411.	65.
3	80.00000	24.00000	13179.	2650.	8483.	17898.	-2070.	9984.	61.
5	120.00000	36.00000	7465.	9091.	8306.	16623.	-67.	8345.	42.
6	20.00000	15.00000	19249.	8462.	13414.	28313.	-602.	14458.	56.

Only the nodes that are used to define plates will be included in the table.

These averages are derived from the stresses in actual triangular plates and in triangular subelements used in the idealization of quadrilateral plates. For this reason stresses may be higher or lower at the nodes than the average plate element stresses shown in the previous Section.

===Fastener forces and stresses===

After the plate data has been reported in the output file, SAFESOLV includes a table with forces and stresses in the fasteners ---if any are present:

(continued)

FASTENER FORCES AND STRESSES

Fastener	I	J	FX	FY	F	Angle	Stress
1	7	22	88.	-1.	88.	-1.	351.
2	8	23	29.	2.	29.	3.	115.
3	9	24	84.	-1.	84.	-1.	334.

Node I corresponds to the first node of the fastener and node J corresponds to the second node.

The FX and FY values represent the fastener load transfer in the x and y directions respectively.

The F and Angle values represent the vector sum of the load transfer components and its line of action relative to the global x-axis.

The Stress value represents the fastener shear stress due to load transfer.

===Node internal forces and reactions===

A summation of all element corner forces and applied nodal loads for each node is presented in this section. The results of this summation show the reaction forces for each node.

NODE INTERNAL FORCES AND REACTIONS

Node	Coord-X	Coord-Y	FX	FY	MZ
1	.00000	.00000	-748. Reaction	134. Reaction	27455. Reaction
2	.00000	96.00000	0.	0.	0.
3	432.00000	96.00000	0.	0.	0.
4	432.00000	.00000	-512. Reaction	226. Reaction	27585. Reaction

The coordinates for each node followed by the force in the x-direction, the force in the y-direction and the moment in the z-direction are presented.

If the model idealization results in a poorly or ill-conditioned matrix, the inaccuracy will be shown as a nonzero value for nodes with no restrained degrees of freedom. Each restrained degree of freedom is indicated by the word Reaction printed after the force value to aid in identifying the reactions. A review of this data is necessary to prove the accuracy of the solution.

===PROGRAM MESSAGES=====

There are three kinds of messages that the MICROSAFE 2-D programs provide:

Informative messages: The ones the programs use to keep you up to date about what they are doing.

Warning messages: The programs inform the user of facts that may be right but, if wrong, may have disastrous consequences.

Error messages: Diagnostics the programs make when they encounter a situation they cannot handle and that requires some changes to be introduced by the user.

The following is a sample of the Section in the manual that describes the error messages:

Message:

ERROR : INCOMPATIBLE TYPE OF NUMERIC ENTRY IN INPUT LINE.
Encountered in line 1066 of file WRNGTYPE.INP.

```
-----|-----
|108 192 199 2.4  0 .020 1 /|
-----|-----
```

Reading properties of plate 108 it was expected to find the index of the third node of the plate - an integer between 1 and 400 - as the fourth entry.

Occurrence: Both the SAFEPLLOT and the SAFESOLV programs.

Explanation: One of the entries in the line is of a type (integer, real, ...) incompatible with the type of the expected data.

The arrows in the error message point to the character responsible for the type change and the following message states the required type for the entry.

Action: Check the input file and replace the entry with the incorrect type.

Execute the program again with the modified input file.

===EXAMPLES=====

Beam With Multiple Supports

This example shows the program in one of its simplest uses. The model, specified in file BEAM.INP, represents a beam of constant properties with fully fixed supports at the left end and a simple support near mid-span. A 500 pound per inch distributed load is applied between the supports and nodal loads are applied at the free end of the cantilevered portion of the beam.

THIS IS FILE BEAM.INP AND WAS USED FOR EXAMPLE #1 IN THE DOCUMENT
This model represents a beam fixed at the left end with a mid-span support.

The left span has distributed loads and the right end has concentrated loads.

```

9 / # nodes
1 / # materials
8 / # beams
0 / # plates
0 / # fasteners
1 / # loaded nodes
4 / # specified displacements
1 0.0 0.0 / nodal coordinates
2 2.5 0.0 /
3 5.0 0.0 /
4 7.5 0.0 /
5 10.0 0.0 /
6 15.0 0.0 /
7 20.0 0.0 /
8 25.0 0.0 /
9 30.0 0.0 /
1 10.3E6 0.33 / material properties
1 1 2 0.5 1.0 1 -500. -500. / beams
2 2 3 0.5 1.0 1 -500. -500. /
3 3 4 0.5 1.0 1 -500. -500. /
4 4 5 0.5 1.0 1 -500. -500. /
5 5 6 0.5 1.0 1 0. 0. /
6 6 7 0.5 1.0 1 0. 0. /
7 7 8 0.5 1.0 1 0. 0. /
8 8 9 0.5 1.0 1 0. 0. /
9 1000. -500. 10000. / applied nodal loads PX PY MZ
1 1 0.0 / specified displacements
1 2 0.0 /
1 3 0.0 /
5 2 0.0 /

```

===Skin Lap-splice With Fasteners=====

The lap-splice model represents two .050 inch thick sheet metal rectangles that are fastened together with three fasteners. The sheet metal parts are defined in separate regions so that they do not overlap and thus the plot of each part does not interfere with the plot of the other part.

Fasteners transfer loads from one node to another according to the relative deflections of the nodes, and the actual locations of the

nodes are ignored. For this reason parts being fastened together may be located anywhere the user desires.

(continued)

July

THIS IS FILE SPLICE.INP : EXAMPLE #2a IN THE MANUAL.
The model represents two pieces of sheet metal,
represented by QUADRILATERAL plates, fastened together
with three rivets. These pieces are positioned
side-by-side to avoid overlapping and allow the plots
to be easier to see.

```
30 / number of nodes
1 / number of materials
0 / number of beams
16 / number of plates
3 / number of fasteners
3 / number of loaded nodes
6 / number of restrained degrees of freedom
1 0.2 0 / node definitions
2 1.2 0 /
3 2.2 0 /
4 3.2 0 /
5 4.2 0 /
6 0.2 1 /
7 1.2 1 /
8 2.2 1 /
9 3.2 1 /
10 4.2 1 /
11 0.2 2 /
12 1.2 2 /
13 2.2 2 /
14 3.2 2 /
15 4.2 2 /
16 0 3 /
17 1 3 /
18 2 3 /
19 3 3 /
20 4 3 /
21 0 4 /
22 1 4 /
23 2 4 /
```

Other example models are defined on the MICROSAFE DISK #1

```
=====
***** MICROSAFE 2-D PURCHASE FORM *****
=====
```

Purchased from: MICROSTRESS Corporation
10950 FOREST AVE SO.
SEATTLE, WASHINGTON, 98178

Federal Tax No. #: 91-1287902
Wash. State Tax #: C 600 572 139
Purchase date: / /

PRODUCT DESCRIPTION	QTY	PRICE EACH	PRICE EXTENDED
=====	===	=====	=====
SAFESOLV and SAFESOLB - solution program		\$50	
SAFEPLLOT - plotting program		\$50	
MICROSAFE PACKAGE - all above programs		\$75	

TOTAL PURCHASE PRICE =

PLEASE MAKE CHECKS PAYABLE TO: MICROSTRESS Corporation

The above amounts include state and local taxes within Washington
state and all postage within the USA. Please include appropriate
postage and instructions for mailing outside of the USA.

Formal documentation is included with all programs shown above.

The undersigned has read the "Usage and Copying of the MICROSAFE 2-D package" and "Limitations of Liability for Use and the Results of Use" sections of the MICROSAFE 2-D Manual Brochure and agrees to abide by these terms and conditions.

Signature

MAILING ADDRESS:

name

street address or P.O. box

city, state and ZIP code

setstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE SETSTR - SUBROUTINE TO MARK THE PHYSICAL END OF A STRING
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

SETSTR is a routine designed to be called from FORTRAN as a subroutine to mark the physical end of a string at the location specified by the input.

Mode of use:

call SETSTR (i,string)

where

i = index to specify the location in the string of the character where the end-of-string mark is to be written.
string = name of the string (variable of type CHARACTER) to be marked.

This routine only checks the index "i" is greater than zero.

*

SUBTTL FORMAL DECLARATIONS

PAGE

cssets SEGMENT 'CODE'
ASSUME CS:cssets

SUBTTL SETSTR - EXECUTABLE CODE

PAGE

PUBLIC SETSTR
SETSTR PROC FAR

PUSH BP
MOV BP,SP
PUSH ds
LDS BX,DWORD PTR [BP+10]
MOV CX,[BX]
DEC CX

; Check positive request.

cmp cx,0
jge inbounds
xor cx,cx

; Mark the physical end of the string.

inbounds:

LDS BX,DWORD PTR [BP+6]
ADD BX,CX
XOR AL,AL
MOV [BX],AL

(continued)

July

```
; Exit.
        POP ds
        MOV SP,BP
        POP BP
        RET 8H

SETSTR  ENDP
cssets  ENDS

END
```

time.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE TIME - SUBROUTINE TO GET THE TIME OF THE DAY
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

```
COMMENT *
TIME is a routine designed to be called from FORTRAN as a subroutine
to access the time of the day from the system.

Mode of use:
        call TIME (hour,minute,second,hundredth)
where
        hour = integer variable containing the hour of the dayth.
        minute = integer variable containing the minute of the hour.
        second = integer variable containing the second of the minute.
        hundredth = integer variable containing the hundredths of
                    the second.
*
```

SUBTTL FORMAL DECLARATIONS
PAGE

```
cstime  SEGMENT 'CODE'
        ASSUME CS:cstime
```

SUBTTL TIME - EXECUTABLE CODE
PAGE

```
PUBLIC  time
time   PROC  FAR

        PUSH  BP
        MOV   BP,SP
        PUSH  DS

; Call the system function.
        xor   ax,ax
        mov   ah,2Ch
        int   21h

; Handle parameters from calling program
        xor   ax,ax
        mov   al,dl
        LDS   BX,DWORD PTR [BP+6]
        MOV   [BX],ax
        mov   al,dh
        LDS   BX,DWORD PTR [BP+10]
        MOV   [BX],ax
        mov   al,cl
        LDS   BX,DWORD PTR [BP+14]
        MOV   [BX],ax
        mov   al,ch
        LDS   BX,DWORD PTR [BP+18]
        MOV   [BX],ax

; Everything done except housekeeping.
exit:
        POP   DS
        MOV   SP,BP
```



```
POP    BP
RET     10H
```

```
time    ENDP
cstime  ENDS
```

```
END
```

```
pakstr.asm
```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

```
TITLE PAKSTR - SUBROUTINE TO PACK A STRING
PAGE ,132
```

```
; (C) Copyright Microstress Corporation 1984, 1985, 1986
```

```
COMMENT *
      PAKSTR is a routine designed to be called from FORTRAN as a subroutine
      to pack a string by eliminating blank characters at both ends.
```

```
Mode of use:
```

```
      call PAKSTR (string)
```

```
where
```

```
      string = name of the string (variable of type CHARACTER) to be
      reset.
```

```
*
```

```
SUBTTL FORMAL DECLARATIONS
PAGE
```

```
cspaks  SEGMENT 'CODE'
        ASSUME  CS:cspaks
```

```
SUBTTL PAKSTR - EXECUTABLE CODE
PAGE
```

```
PUBLIC  PAKSTR
PAKSTR PROC FAR
```

```
    PUSH BP
    MOV BP,SP
    PUSH ds
    LDS SI,DWORD PTR [BP+6]
    mov DI,si
```

```
; Scan for the first non blank character
scanonblank:
```

```
    mov al,[si]
    inc si
    cmp al,32
    jz scanonblank
```

```
; Move characters until any end-of-string mark is found.
```

```
    dec si
    xor cx,cx
```

```
scanend:
```

```
    mov al,[si]
    cmp al,0
    jz scanback
    cmp al,6
    jz scanback
    mov [di],al
    inc si
    inc di
    inc cx
    jmp scanend
```

```
; Move back until a non-blank character or the beginning of the string is found.
```

```
scanback:
```

```
    cmp cx,0
    je logical
    dec cx
```

(continued)

July

```
    dec di
    mov al,[di]
    cmp al,32
    jz scanback
    inc di
; Add a logical end-of-string mark if necessary.
logical:
    cmp di,si
    jz exit
    mov al,6
    mov [di],al
; Exit
exit:
    POP ds
    MOV SP,BP
    POP BP
    RET 4

PAKSTR ENDP
cspaks ENDS

END
```

confirm.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE CONFIRM - SUBROUTINE TO CONFIRM AN OPERATION BY PRESSING ANY KEY
PAGE ,132

; (C) Copyright Microstress Corporation 1985, 1986

COMMENT *
 Mode of use: call CONFIRM
 *

SUBTTL FORMAL DECLARATIONS
PAGE

csconf SEGMENT 'CODE'
 ASSUME CS:csconf

SUBTTL CONFIRM - EXECUTABLE CODE
PAGE

PUBLIC confirm
confirm PROC FAR

```
    PUSH    BP
    MOV     BP,SP
    mov     al,8
    mov     ah,0Ch
    int     21h
    MOV     SP,BP
    POP     BP
    RET
```

confirm ENDP

csconf ENDS

END

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE PACER - SUBROUTINE TO PACE THE EXECUTION OF A PROGRAM
PAGE .132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

PACER is a routine designed to be called from FORTRAN as a subroutine to plot a symbol in the screen to show a program is running.

Mode of use:

every time is required to call PACER
* show the pace.

SUBTTL FORMAL DECLARATIONS
PAGE

```
CSPACE SEGMENT 'CODE'
ASSUME CS:CSPACE
```

SUBTTL PACER - EXECUTABLE CODE
PAGE

PUBLIC PACER
PACER PROC FAR

```

PUSH BP
MOV BP,SP
mov ah,10
xor bh,bh
mov cx,1
mov al,176
int 10h
mov ah,3
xor bh,bh
int 10h
mov ah,2
inc dl
int 10h
MOV SP,BP
POP BP
RET

```

```
PACER      ENDP
CSPACE    ENDS
```

END

logpsl.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE LOGPSL - SUBROUTINE TO PUT IN THE SCREEN THE LOGO OF THE PROGRAM
PAGE .132

; (C) Copyright Fernando G. Loygorri 1984, 1985, 1986

```
COMMENT *
      Mode of use:
                                call logpsl
      *
```

SUBTTL FORMAL DECLARATIONS
PAGE

(continued)


```

mov     ah,2
int     10h
MOV     SP,BP
POP     BP
RET

```

```
logpsl  ENDP
```

```
cslogd  ENDS
```

```
END
```

```
intsgn.asm
```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE INTSGN - FUNCTION TO DETERMINE THE SIGN OF AN INTEGER
PAGE ,132

; (C) Copyright Microstress Corporation 1985, 1986

```

COMMENT *
  INTSGN is a routine designed to be called from FORTRAN as a function
  to return the sign of an integer.

  Mode of use:
      sign = INTSGN (integer)
  where
      sign = value returned by the function, the integer sign.
      integer = name of the integer (variable of type INTEGER-2) whose
                sign is requested.
  *

```

SUBTTL FORMAL DECLARATIONS
PAGE

```

csisgn  SEGMENT 'CODE'
        ASSUME  CS:csisgn

```

SUBTTL INTSGN - EXECUTABLE CODE
PAGE

```

PUBLIC  INTSGN
INTSGN  PROC  FAR

        PUSH BP
        MOV BP,SP
        PUSH ds
        LDS BX,DWORD PTR [BP+6]
        mov cx,[bx]
        cmp cx,0
        jg  positive
        jl  negative
        xor ax,ax
        jmp exit

positive:
        mov ax,1
        jmp exit

negative:
        mov ax,0ffffh
; Value is returned in AX.
exit:
        POP ds
        MOV SP,BP
        POP BP
        RET 4H

```

```

INTSGN  ENDP
csisgn  ENDS

```

```
END
```

(continued)

constr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE CONSTR - SUBROUTINE TO CONCATENATE TWO STRINGS
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *
CONSTR is a routine designed to be called from FORTRAN as a subroutine to concatenate two strings.

Mode of use:

call CONSTR (deststr,sourstr)

where

deststr = destination string name.

sourstr = source string name to be concatenated to "deststr".

*

SUBTTL FORMAL DECLARATIONS
PAGE

cscons SEGMENT 'CODE'
ASSUME CS:cscons

SUBTTL CONSTR - EXECUTABLE CODE
PAGE

PUBLIC CONSTR
CONSTR PROC FAR

PUSH BP
MOV BP,SP
PUSH DS
PUSH ES

; Locate end of the destination string (logical or physical)
LES di,DWORD PTR [BP+10]
dec di

scanend1:
inc di
mov al,es:[di]
cmp al,0
je exit
cmp al,6
jne scanend1
mov bx,di

; Locate the physical end of the destination string.
scanend2:

inc bx
mov al,es:[bx]
cmp al,0
jne scanend2
sub bx,di
mov dx,bx

; Locate end of the source string (logical or physical)
LDS si,DWORD PTR [BP+6]
mov bx,si
xor cx,cx

scanendsource:
mov al,[bx]
cmp al,0
je foundend
cmp al,6
je foundend
inc bx
inc cx
jmp scanendsource

; Check if there is enough room.

foundend:
cmp dx,cx
jge copy


```

        mov cx,dx
; Tack the source string at the end of the destination string.
copy:   rep movsb
; Add a logical end-of-string mark if not physical end already.
        mov al,es:[di]
        cmp al,0
        jz exit
        mov al,6
        mov es:[di],al
; Everything done except housekeeping.
exit:
        POP ES
        POP DS
        MOV SP,BP
        POP BP
        RET 8

CONSTR  ENDP
cscons  ENDS

END

```

```
lenstr.asm
```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

```

TITLE LENSTR - FUNCTION TO DETERMINE STRING LENGTH
PAGE ,132

```

```
; (C) Copyright Microstress Corporation 1984, 1985, 1986
```

```

COMMENT *
LENSTR is a routine designed to be called from FORTRAN as a function
to return the length of a string.

Mode of use:
                                length = LENSTR (string)
where
        length = 2-byte integer returned, the string length.
        string = name of the string (variable of type CHARACTER) whose
                  length is requested.
*

```

```

SUBTTL FORMAL DECLARATIONS
PAGE

```

```

cslens  SEGMENT 'CODE'
        ASSUME CS:cslens

```

```

SUBTTL LENSTR - EXECUTABLE CODE
PAGE

```

```

PUBLIC lenstr
lenstr  PROC FAR

```

```

        PUSH    BP
        MOV     BP,SP
        PUSH    ds
        LDS     BX,DWORD PTR [BP+6]
        xor     ax,ax

scanend:
        mov     cl,[bx]
        cmp     cl,0
        je      exit
        cmp     cl,6
        je      exit
        inc     bx
        inc     ax
        jmp     scanend
; Value is returned in AX.
exit:

```

(continued)

July

```
POP    ds
MOV    SP,BP
POP    BP
RET    4
```

```
lenstr ENDP
cslens ENDS
```

END

lwcstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE LWCSTR - SUBROUTINE TO PUT A STRING IN LOWERCASE
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *
LWCSTR is a routine designed to be called from FORTRAN as a subroutine to set all the alphabetic characters to lowercase.

Mode of use:

call LWCSTR (string)

where

string = name of the string (variable of type CHARACTER) to be converted to lowercase.

*

SUBTTL FORMAL DECLARATIONS
PAGE

```
cslwcs SEGMENT 'CODE'
ASSUME CS:cslwcs
```

SUBTTL LWCSTR - EXECUTABLE CODE
PAGE

```
PUBLIC lwcstr
lwcstr PROC FAR
```

```
PUSH    bp
MOV     bp,sp
PUSH    ds
LDS     BX,DWORD PTR [BP+6]
```

character:

```
CMP     BYTE PTR [bx],0
je      exit
CMP     BYTE PTR [bx],6
je      exit
CMP     BYTE PTR [bx],'A'
JB      next
CMP     BYTE PTR [bx],'Z'
JA      next
OR      BYTE PTR [bx],32
```

next:

```
INC     bx
jmp     character
```

exit:

```
pop     ds
MOV     sp,bp
POP     bp
RET     4
```

```
lwcstr ENDP
cslwcs ENDS
```

END

modstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE MODSTR - SUBROUTINE TO MODIFY A STRING BY REPLACING A CHARACTER
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *
MODSTR is a routine designed to be called from FORTRAN as a subroutine
to modify a string by replacing a character in it.

Mode of use:

call MODSTR (deststr,position,asciic)

where

deststr = destination string name.

position = character in "deststr" to be replaced.

asciic = ASCII code of the character to be put in the string.

*

SUBTTL FORMAL DECLARATIONS
PAGE

csmode SEGMENT 'CODE'
ASSUME CS:csmode

SUBTTL MODSTR - EXECUTABLE CODE
PAGE

PUBLIC modstr
modstr PROC FAR

PUSH BP

MOV BP,SP

PUSH DS

; Handle parameters from calling program

LDS BX,DWORD PTR [BP+6]

MOV ax,DS:[BX]

LDS BX,DWORD PTR [BP+10]

MOV cx,DS:[BX]

LDS bx,DWORD PTR [BP+14]

; Check positive request for location in destination string.

cmp cx,0

jle exit

dec cx

add bx,cx

mov [bx],al

; Everything done except housekeeping.

exit:

POP DS

MOV SP,BP

POP BP

RET 0CH

modstr ENDP
csmode ENDS

END

filstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE FILSTR - SUBROUTINE TO FILL A STRING WITH A GIVEN CHARACTER
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985

(continued)

COMMENT *

FILSTR is a routine designed to be called from FORTRAN as a subroutine to fill a string with a character specified as a parameter.

Mode of use:

call FILSTR (i,string)

where

i = ASCII code of the character to be used to fill the string.
 string = name of the string (variable of type CHARACTER) to be reset.

*

SUBTTL FORMAL DECLARATIONS
 PAGE

csfils SEGMENT 'CODE'
 ASSUME CS:csfils

SUBTTL FILSTR - EXECUTABLE CODE
 PAGE

PUBLIC FILSTR
 FILSTR PROC FAR

PUSH BP
 MOV BP,SP
 PUSH ds
 LDS BX,DWORD PTR [BP+10]
 mov ah,[bx]
 LDS BX,DWORD PTR [BP+6]

; Write character until an end-of-string mark is found.
 scanend:

mov al,[bx]
 cmp al,0
 jz exit
 cmp al,6
 jz exit
 mov [bx],ah
 inc bx
 jmp scanend

; Exit
 exit:

POP ds
 MOV SP,BP
 POP BP
 RET 8

FILSTR ENDP
 csfils ENDS

END

sizstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando
 G. Loygorri. July, page 199.

TITLE SIZSTR - FUNCTION TO DETERMINE STRING SIZE
 PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

SIZSTR is a routine designed to be called from FORTRAN as a function to return the size of a string.

Mode of use:

size = SIZSTR (string)

where

size = 2-byte integer returned by the function, the string size.
 string = name of the string (variable of type CHARACTER) whose
 length is requested.

*

SUBTTL FORMAL DECLARATIONS
PAGE

cssize SEGMENT 'CODE'
ASSUME CS:cssize

SUBTTL SIZSTR - EXECUTABLE CODE
PAGE

PUBLIC sizstr
sizstr PROC FAR

PUSH BP
MOV BP,SP
PUSH ds
LDS BX,DWORD PTR [BP+6]
xor ax,ax

scanend:
mov cl,[bx]
inc ax
cmp cl,0
je exit
inc bx
jmp scanend

; Value is returned in AX.

exit:
POP ds
MOV SP,BP
POP BP
RET 4h

sizstr ENDP
cssize ENDS

END

endstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE ENDSTR - SUBROUTINE TO MARK THE LOGICAL END OF A STRING
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

ENDSTR is a routine designed to be called from FORTRAN as a subroutine
to mark the logical end of a string at the location specified by the
input.

Mode of use:

call ENDSTR (i,string)

where

i = index to specify the location in the string of the character
where the logical-end-of-string mark is to be written.
string = name of the string (variable of type CHARACTER) to be
marked.

*

SUBTTL FORMAL DECLARATIONS
PAGE

csends SEGMENT 'CODE'
ASSUME CS:csends

SUBTTL ENDSTR - EXECUTABLE CODE
PAGE

(continued)

July

```
PUBLIC ENDSTR
ENDSTR PROC FAR

    PUSH BP
    MOV BP,SP
    PUSH ds
    LDS BX,DWORD PTR [BP+10]
    MOV CX,[BX]
    DEC CX
; Check positive request.
    cmp cx,0
    jge inbounds
    xor cx,cx
; Check the logical mark is not beyond the physical mark.
inbounds:
    LDS BX,DWORD PTR [BP+6]
scanend:
    mov al,[bx]
    cmp al,0
    jz exit
    cmp cx,0
    jz mark
    dec cx
    inc bx
    jmp scanend
; Mark the logical end of the string.
mark:
    mov AL,6
    MOV [BX],AL
; Exit.
exit:
    POP ds
    MOV SP,BP
    POP BP
    RET 8H

ENDSTR ENDP
csends ENDS

END
```

delfil.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE DELFIL - SUBROUTINE TO DELETE A FILE
PAGE ,132

; (C) Copyright Microstress Corporation 1985, 1986

COMMENT *
 DELFIL is a routine designed to be called from FORTRAN as a subroutine
 to delete a file specified with drive (optional), path (optional), name
 and extension in an ASCII string.

Mode of use:

call DELFIL (fname)

where

fname = name of the file (variable of type CHARACTER) to be
deleted.

*

SUBTTL FORMAL DECLARATIONS
PAGE

csdelf SEGMENT 'CODE'
 ASSUME CS:csdelf

SUBTTL DELFIL - EXECUTABLE CODE
PAGE


```

PUBLIC delfil
delfil PROC FAR

    PUSH BP
    MOV BP,SP
    PUSH ds
    LDS dx,DWORD PTR [BP+6]
; Use Function Call from DOS 2.0.
    mov ah,41h
    int 21h
    POP ds
    MOV SP,BP
    POP BP
    RET 4

delfil ENDP
csdelf ENDS

END

```

```
locstr.asm
```

```
"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.
```

```

TITLE LOCSTR - SUBROUTINE TO LOCATE A STRING INSIDE ANOTHER STRING
PAGE ,132

```

```
; (C) Copyright Microstress Corporation 1984,1985, 1986
```

```

COMMENT *
LOCSTR is a routine designed to be called from FORTRAN as a function
to locate a string inside another string.

Mode of use:
                                pos = LOCSTR (i,string1,string2)
where
    i = location where the search in the destination string is to
        start.
    string1 = name of the destination string (variable of type
        CHARACTER) where the source string is to be searched.
    string2 = name of the source string (variable of type CHARACTER)
        to be located inside the destination string.
*

```

```

SUBTTL FORMAL DECLARATIONS
PAGE

```

```

cslocs SEGMENT 'CODE'
ASSUME CS:cslocs

```

```

SUBTTL LOCSTR - EXECUTABLE CODE
PAGE

```

```

PUBLIC locstr
locstr PROC FAR

    PUSH BP
    MOV BP,SP
    PUSH DS
    PUSH ES

; Handle parameters from calling program
    LES BX,DWORD PTR [BP+14]
    MOV CX,ES:[BX]

; Check positive request.
    push cx
    cmp cx,0
    jle outbounds
    LES DX,DWORD PTR [BP+10]
    mov bx,dx

scanend:

```

(continued)

```

        mov     al,es:[bx]
        cmp     al,0
        je      outbounds
        cmp     al,6
        je      outbounds
        inc     bx
        loop    scanend
; The request is legitimate.
        pop     cx
        LDS     BX,DWORD PTR [BP+6]
        DEC     CX
        push    cx
        ADD     DX,CX
; Determine the length of the destination string.
        xor     si,si
        mov     di,dx
scanendest:
        mov     al,es:[di]
        inc     si
        inc     di
        cmp     al,0
        jz      end_dest
        cmp     al,6
        jnz     scanendest
end_dest:
        mov     cx,si
        push    cx
; Determine the length of the source string.
        xor     si,si
        mov     di,bx
scanensour:
        mov     al,ds:[di]
        inc     si
        inc     di
        cmp     al,0
        jz      end_sour
        cmp     al,6
        jnz     scanensour
end_sour:
        xchg    bx,si
        push    si
; Locate source string in destination string.
scandest:
        mov     di,dx
        pop     si
        push    si
        xor     ah,ah
        mov     al,ds:[si]
        cld
        repnz   scasb
        jcxz    failure
; First character of source is found in destination, check the rest.
        mov     dx,di
        dec     di
        push    cx
        mov     cx,bx
        repz    cmpsb
        jcxz    success
; No complete match, proceed search in destination string
        pop     cx
        jmp     scandest
; The index for the request was out of bounds.
outbounds:
        pop     cx
        xor     ax,ax
        jmp     exit
; No match was found, return zero.
failure:
        pop     si
        pop     cx
        pop     cx
        xor     ax,ax
        jmp     exit
; A complete match was found, return location of first character.
success:

```



```

        pop     cx
        pop     si
        pop     ax
        sub     ax,cx
        pop     cx
        add     ax,cx
; Everything done except housekeeping.
exit:
        POP     ES
        POP     DS
        MOV     SP,BP
        POP     BP
        RET     0Ch

locstr  ENDP
cslocs  ENDS

END

```

```
resstr.asm
```

```

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

```

```

TITLE RESSTR - SUBROUTINE TO RESET A STRING TO FORTRAN STANDARDS
PAGE ,132

```

```
; (C) Copyright Microstress Corporation 1984, 1985
```

```

COMMENT *
RESSTR is a routine designed to be called from FORTRAN as a subroutine
to reset a string to FORTRAN standards by removing end-of-string marks
and filling the string with blanks.

```

```
Mode of use:
```

```
        call RESSTR (string)
```

```
where
```

```
        string = name of the string (variable of type CHARACTER) to be
                reset.
```

```
*
```

```

SUBTTL FORMAL DECLARATIONS
PAGE

```

```

csress  SEGMENT 'CODE'
        ASSUME  CS:csress

```

```

SUBTTL RESSTR - EXECUTABLE CODE
PAGE

```

```

PUBLIC  RESSTR
RESSTR  PROC  FAR

```

```

        PUSH BP
        MOV BP,SP
        PUSH ds
        LDS BX,DWORD PTR [BP+6]
        mov ah,32

```

```
; Scan for either end-of-string mark.
```

```

scanend:
        mov al,[bx]
        cmp al,0
        jz physical
        cmp al,6
        jz logical
        inc bx
        jmp scanend

```

```
; Fill with blanks between logical and physical marks.
```

```

logical:
        MOV [BX],ah
        inc bx
        mov al,[bx]

```

(continued)

July

```
        cmp al,0
        jnz logical
; Remove the physical end of string mark
physical:
        MOV [BX],ah
; Exit
        POP ds
        MOV SP,BP
        POP BP
        RET 4

RESSTR ENDP
csress ENDS

END
```

defdrv.asm

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

TITLE DEFDRV - SUBROUTINE TO GET OR SET
THE DEFAULT DISK DRIVE

PAGE ,132

; (C) Copyright Microstress
Corporation 1984, 1985, 1986

COMMENT *
 Mode of use: call DEFDRV (oper,unit)
 where
oper = integer to specify get (=0) or set (=1).
unit = integer for drive number (0=A,1=B,...).
 *

SUBTTL FORMAL DECLARATIONS
PAGE

csdefd SEGMENT 'CODE'
 ASSUME CS:csdefd

SUBTTL DEFDRV - EXECUTABLE CODE
PAGE

PUBLIC DEFDRV
DEFDRV PROC FAR

```
        PUSH    BP
        MOV     BP,SP
        push    ds
        MOV     BX,[BP]+10
        MOV     ax,[BX]
        cmp     ax,0
        jnz     set
        MOV     ah,19h
        int     21h
        mov     bx,[bp]+6
        xor     ah,ah
        mov     [bx],ax
        jmp     exit

set:
        mov     bx,[bp]+6
        mov     dx,[bx]
        mov     ah,0Eh
        int     21h

exit:
        pop     ds
        MOV     SP,BP
        POP     BP
        RET     8
```



```
DEFDRV  endp
csdefd  ENDS
END
```

```
dskfre.asm
```

"Structural Analysis," by Robert W. Johnson and Fernando G. Loygorri. July, page 199.

```
TITLE DSKFRE - SUBROUTINE TO GET THE DISK FREE SPACE AVAILABLE.
PAGE ,132
```

```
; (C) Copyright Microstress Corporation 1985, 1986
```

```
COMMENT *
      DSKFRE is a routine designed to be called from FORTRAN as a subroutine
      to determine the available disk free space.

      Mode of use:
              call DSKFRE (drive,nscl,nfcl,ntcl,nbys)
      where
              drive = integer variable containing the drive number (0=default,
                        1=A, etc...)
              nscl  = integer variable containing the number of sectors per
                        cluster or an error flag (=FFFFh).
              nfcl  = integer variable containing the number of free clusters.
              ntcl  = integer variable containing the total number of clusters.
              nbys  = integer variable containing the number of bytes per
                        sector.
```

```
*
```

```
SUBTTL FORMAL DECLARATIONS
PAGE
```

```
csdskf  SEGMENT 'CODE'
        ASSUME CS:csdskf
```

```
SUBTTL DSKFRE - EXECUTABLE CODE
PAGE
```

```
PUBLIC  DSKFRE
DSKFRE PROC FAR
```

```
        PUSH BP
        MOV BP,SP
        PUSH DS
; Get the drive number.
        LDS BX,DWORD PTR [BP+22]
        MOV dx,[bx]
; Call the system function.
        mov ah,36h
        int 21h
; Handle parameters from calling program
        cmp ax,0FFFFh
        je  exit
        LDS si,DWORD PTR [BP+6]
        MOV [si],cx
        LDS si,DWORD PTR [BP+10]
        MOV [si],dx
        LDS si,DWORD PTR [BP+14]
        MOV [si],bx
; Everything done except housekeeping.
exit:
        LDS si,DWORD PTR [BP+18]
        MOV [si],ax
        POP DS
        MOV SP,BP
        POP BP
        RET 14H
```

(continued)

July

DSKFRE ENDP
csdskf ENDS

END

sizfil.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE SIZFIL - FUNCTION TO DETERMINE FILE SIZE
PAGE ,132

; (C) Copyright Microstress Corporation 1985, 1986

COMMENT *
SIZFIL is a routine designed to be called from FORTRAN as a function
to return the size in bytes of the file specified by the input.

Mode of use:

size = SIZFIL (filename)

where

size = 4-byte integer returned with the file size.
filename = name of the file (variable of type CHARACTER) whose
size is requested.

*

SUBTTL FORMAL DECLARATIONS
PAGE

cssizf SEGMENT 'CODE'
ASSUME CS:cssizf

SUBTTL SIZFIL - EXECUTABLE CODE
PAGE

PUBLIC SIZFIL
SIZFIL PROC FAR

PUSH BP
MOV BP,SP
PUSH ds
LDS DX,DWORD PTR [BP+6]
; Find first (and only) matching file
xor cx,cx
mov ah,4Eh
int 21h
; Get the current Disk Transfer Address
mov ah,2Fh
int 21h
add bx,26
mov ax,es:[bx]
mov dx,es:[bx]+2
; Exit
POP ds
MOV SP,BP
POP BP
RET 4

SIZFIL ENDP
cssizf ENDS

END

lisptest.doc

"BYSO Lisp and Waltz Lisp," by William Wong. July, Page 293

"BYSO Lisp Benchmark 1-4-86 WGW"

"Test Loop"

```
(defun loop-test (fn limit)
  (do ((i 1 (+ i 1)))
      ((= i limit))
    (fn) ))
```

```
(defun dummy ())
```

"CONS Test"

```
(setq cons-a nil)
```

```
(defun cons-test () (cons cons-a cons-a))
```

"Integer Addition Test"

```
(setq add-a 1 add-b 2)
```

```
(defun add-test () (+ add-a add-b))
```

"Integer Multiplication Test"

```
(setq multiply-a 1 multiply-b 2)
```

```
(defun multiply-test () (* multiply-a multiply-b))
```

"Assignment Test"

```
(setq assign-a '(1 2 3))
```

```
(defun assign-test () (setq assign-a assign-a))
```

"List Indexing Test"

```
(setq list-index-list '())
```

```
(do ((i 1 (+ i 1)))
    ((= i 128))
  (setq list-index-list (cons i list-index-list)))
```

```
(defun list-index () (nth 120 list-index-list))
```

"Vector Index Test"

```
(setq vector-test-array (array 'sexpr 128))
```

```
(defun vector-index () (aref vector-test-array 120))
```

"String Index Test"

```
(setq string-test-array (array 'char 128))
```

```
(defun string-index () (aref string-test-array 120))
```

"Write test creates a new file and writes 64 kbytes to it."

(continued)

```

( defun write-test ()
  ( do-write-test ( open 'b:test )
                    512
                    ( array 'char 128 )
  )
)

( defun do-write-test ( file records buffer )
  ( do ( ( ( zerop ( setq records ( - records 1 ) ) ) ( close file ) )
          ( princ buffer file )
  )
)

; Waltz Lisp Benchmark          1-4-86 WGW
;
; Test Loop

(def loop-test (lambda (fn limit)
  (do ((i 1 (+ i 1)))
      ((equal i limit))
      (fn) ) ))

(def dummy (lambda ()))

; CONS Test

(setq cons-a nil)

(def cons-test (lambda () (cons cons-a cons-a)))

; Integer Addition Test

(setq add-a 1)
(setq add-b 2)

(def add-test (lambda () (+ add-a add-b)))

; Integer Multiplication Test

(setq multiply-a 1)
(setq multiply-b 2)

(def multiply-test (lambda () (* multiply-a multiply-b)))

; Assignment Test

(setq assign-a '(1 2 3))

(def assign-test (lambda () (setq assign-a assign-a)))

; List Indexing Test

(setq list-index-list '())

(do ((i 0 (+ i 1)))
    ((equal i 128))
    (setq list-index-list (cons i list-index-list)) )

(def list-index (lambda () (nth 120 list-index-list)))

; Vector Index Test (Arrays Not Supported)

; String Index Test

(setq string-test-array "" )

```



```

(do ((i 0 (+ i 1)))
  ((equal i 128))
  (setq string-test-array (cat "1" string-test-array)) )

(def string-index (lambda () (substring string-test-array 120 120)))

;; Write test creates a new file and writes 64 kbytes to it.
(def write-test (lambda ()
  (do-write-test ( outfile "b:test" )
    512
    string-test-array ) ))

(def do-write-test (lambda (file records buffer)
  (do ()
    (( zerop ( setq records ( - records 1 ))) ( close file ))
    ( princ buffer file ) ) ))

;; Golden Common Lisp Benchmark 1-4-86 WGW

;; Test Loop
(defun loop-test (fn limit)
  (do (( i 1 ( + i 1 )))
    ((= i limit))
    (apply fn nil) ) )

(defun dummy () )

;; CONS Test
(setq cons-a nil)
(defun cons-test () (cons cons-a cons-a))

;; Integer Addition Test
(setq add-a 1 add-b 2)
(defun add-test () (+ add-a add-b))

;; Integer Multiplication Test
(setq multiply-a 1 multiply-b 2)
(defun multiply-test () (* multiply-a multiply-b))

;; Floating Point Addition Test
(setq fp-add-a 1.2 fp-add-b 234324.3)
(defun fp-add-test () (+ fp-add-a fp-add-b))

;; Floating Point Multiplication Test
(setq fp-multiply-a 1.2 fp-multiply-b 234324.3)
(defun fp-multiply-test () (* fp-multiply-a fp-multiply-b))

;; Assignment Test
(setq assign-a '(1 2 3))
(defun assign-test () (setq assign-a assign-a))

```

(continued)

July

;; List Indexing Test

```
(setq list-index-list '())  
(do ((i 1 (+ i 1)))  
    ((= i 128))  
    (setq list-index-list (cons i list-index-list)))  
(defun list-index () (nth 120 list-index-list))
```

;; Vector Index Test

```
(setq vector-test-array (make-array 128 :initial-element nil))  
(defun vector-index () (aref vector-test-array 120))
```

;; String Index Test

```
(setq string-test-array  
  (make-array 128 :element-type 'string-char :initial-element 32))  
(defun string-index () (aref string-test-array 120))
```

"Write test creates a new file and writes 64 kbytes to it."

```
(defun write-test ()  
  (do-write-test (open "b:test" :direction ':output)  
                  512  
                  (make-array 128 :element-type 'string-char)  
  )  
)  
  
(defun do-write-test (file records buffer)  
  (do ()  
    ((zerop (setq records (- records 1))) (close file))  
    (princ buffer file))  
  )  
)
```

pcrack.bas

"Stress Analysis," by D. Lee Petersen and Steven L.
Crocu. July, page 219.

```
1000 REM -----  
1010 REM PROGRAM PCRAK.BAS IN MICROSOFT BASIC  
1020 REM -----  
1030 REM          PROGRAM USAGE NOTES  
1040 REM          (1) USE CONSISTENT PHYSICAL UNITS  
1050 REM          (2) NUMBER OF ELEMENTS <= 30  
1060 REM -----  
1070 DEFINT I-N  
1080 OPTION BASE 1  
1090 DIM A(30,30),DY(30)  
1100 REM ----- READ BASIC MODEL PARAMETERS  
1110 INPUT "CRACK PRESSURE, CRACK LENGTH, NUMBER OF ELEMENTS";P,CL,N  
1120 W=CL/(2!*N)  
1130 INPUT "MATERIAL PROPERTIES - G,PR";G,PR  
1140 REM ----- SET CONSTANTS  
1150 PI=3.14159  
1160 CON=-G/(PI*W*(1!-PR))  
1170 REM ----- GAUSS-SEIDEL ITERATION PARAMETERS  
1180 TOL= .00001  
1190 ITMAX=2*N  
1200 OMEGA=1.3  
1210 REM ----- INITIALIZE DY AND COMPUTE A MATRIX  
1220 FOR I=1 TO N  
1230   DY(I)=0!  
1240   FOR J=1 TO N
```



```

1250      A(I,J)=CON/(4!*(J-I)^2 - 1!)
1260      NEXT J
1270 NEXT I
1280 REM ----- SOLVE EQUATIONS BY GAUSS-SEIDEL ITERATION
1290 FOR NUM = 1 TO ITMAX
1300     ERRMAX=0!
1310     FOR I=1 TO N
1320         TEMP=0!
1330         FOR J=1 TO N
1340             TEMP=TEMP+A(I,J)*DY(J)
1350         NEXT J
1360         TEMP=(P-TEMP)/A(I,I)
1370         DY(I)=DY(I)+OMEGA*TEMP
1380         ERRI=ABS(TEMP)
1390         IF ERRI > ERRMAX THEN ERRMAX=ERRI
1400     NEXT I
1410     IF ERRMAX <= TOL THEN GOTO 1460
1420     PRINT USING"ITERATION, MAXIMUM ITERATE ## #.#####";NUM;ERRMAX
1430 NEXT NUM
1440 PRINT "ITERATION PROCESS DID NOT CONVERGE AFTER";ITMAX;" ITERATIONS!"
1450 STOP
1460 REM ----- PRINT RESULTS
1470 PRINT "ELEM CRACK OPENING CRACK OPENING COMPUTED STRESS"
1480 PRINT "      (NUMERICAL)      (ANALYTICAL)"
1490 DELB=CL/N
1500 X=-(CL+DELB)/2!
1510 B=CL/2
1520 CON=2!*(1!-PR)*P*B/G
1530 FOR I = 1 TO N
1540     SIGYY=0!
1550     FOR J = 1 TO N
1560         SIGYY=SIGYY+A(I,J)*DY(J)
1570     NEXT J
1580     X=X+DELB
1590     DD=CON*SQR(1!-(X^2/B^2))
1600 PRINT USING" ##      ##.####      ##.####      #####.#";I;DY(I);DD;SIGYY
1610 NEXT I
1620 END

```

msp.pas

"A Material Selection Program," by Brother Tom Sawyer and Michael Pecht. July, page 235.

```

PROGRAM MSP;
{ Material selection program }
{ Major sections: Utilities, Filework, Newbase, Addtobase and Searchbase }
{ BTS 04/10/85 major file handling rev. 9/15/85 tuning etc. 1/20/86 }

{$B-}      (*compiler directive for direct read*)

```

```

CONST
maxattributes = 31;
maxqualifiers = 10;
maxalablen = 20;
maxqlablen = 10;
maxitems = 300;
maxroot = 6;

TYPE
attrnum = 0..maxattributes;
itemnum = 0..maxitems;
qualnum = 0..maxqualifiers;
flag = -1..1;
attrlab = STRING[maxalablen];
quallab = STRING[maxqlablen];
itemlab = STRING[maxalablen];
qualifiers = ARRAY[qualnum] OF quallab;
attrdata = (NUL,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10);
itemdata = ARRAY [attrnum] OF attrdata;
attrrec = RECORD
    atnum : attrnum;
    atname : attrlab;

```

(continued)

```

    qlnum : qualnum;
    quals : qualifiers
END;
itemrec = RECORD
    fid : CHAR;
    inum : INTEGER;
    iname : itemlab;
    idata : itemdata
END;
rankitemrec = RECORD
    rating : INTEGER;
    item : itemrec
END;
rootname = STRING[maxroot];
finamrec = RECORD
    id : CHAR;
    rtnam: rootname;
    name : itemlab
END;
Trackrec = RECORD
    nofitems : INTEGER;
    atnum : attrnum;
    atrbute : attrdata;
    selection : CHAR;
    atname : attrlab;
    qualname : quallab
END;
markarray = ARRAY[attrnum] OF BOOLEAN;
attrtable = ARRAY[attrnum] OF attrrec;
Itemtable = ARRAY[itemnum] OF Itemrec;
ranktable = ARRAY[itemnum] OF rankitemrec;
afile = FILE OF attrrec;
ifile = FILE OF itemrec;
Nfile = FILE OF finamrec;
Tfile = FILE OF Trackrec;
stri = string[80];

```

```

VAR
    exit : BOOLEAN;
    ch:CHAR;
    firoot : rootname;
    finame : itemlab;
    valid : STRING[2];
        {Utilities Procedures}

```

```

PROCEDURE Center(astring:stri);    {center a string on the screen}

```

```

VAR
    indent : INTEGER;
BEGIN
    indent := (80 - LENGTH(astring)) DIV 2;
    Writeln('':indent,astring)
END;

```

```

FUNCTION Spa(indent:INTEGER):CHAR;    {insert spaces in a write(ln) statement}

```

```

BEGIN
    indent := indent - 1;
    WRITE('':indent);
    Spa := ' ';
END;

```

```

PROCEDURE Rdupcase(VAR ch:CHAR);    {read and capitalize a character}

```

```

BEGIN
    READ(ch);
    IF ch IN ['a'..'z'] THEN
        ch := CHR(ORD(ch) - 32)
    END;

```

```

PROCEDURE Fixa (lstr:stri;VAR alab:attrlab);
VAR
    {Change a string into an attribute label}

```

```

J:INTEGER;
BEGIN
    alab := '';
    FOR J := 1 TO maxalablen DO
        alab := CONCAT(alab,' ');
    END;

```



```

FOR J := 1 TO LENGTH(lstr) DO
  alab[J] := lstr[J];
END; {Fixa}

      {File handling section}

PROCEDURE Getfilename(VAR firoot:rootname; VAR finame:itemlab;
                     VAR choice:CHAR; VAR exit:BOOLEAN);
  {Create and check new file names, show available data bases}
  {firoot is the root name of the data base files}
  {exit is true if user chooses main menu or files don't exist}
  {choice may be changed if no AT or IM files are present}
VAR
  chx : CHAR;
  nfrec : finamrec; {global to Proc. Getfilename}
  tempstr : stri;
  nffile : Nfile;

PROCEDURE Showbases(VAR chx:CHAR);
BEGIN
  GOTOXY(1,10);
  Center('List of current data base names:');
  WRITELN;
  RESET(nffile);
  WHILE NOT (EOF(nffile)) DO
    BEGIN
      READ(nffile,nfrec);
      WITH nfrec DO
        IF id > '@' THEN
          BEGIN
            WRITELN(Spa(25),id,': ',name);
            chx := id;
          END
        END {while}
      END; {Showbases}

PROCEDURE Addtosearch(VAR firoot:rootname; VAR finame:itemlab;
                     VAR exit:BOOLEAN; chx:CHAR);
VAR
  found : BOOLEAN;
  ch : CHAR;
BEGIN
  found := FALSE;
  REPEAT
    REPEAT
      GOTOXY(1,21);
      Center('Please type in the LETTER of your choice. ');
      Center('Press RETURN to EXIT. ');
      Rdupcase(ch)
    UNTIL (ch IN ['A'..'CHX']) OR (eoln);
    IF eoln THEN
      exit := TRUE
    ELSE
      BEGIN
        RESET(nffile);
        WHILE NOT (EOF(nffile)) DO
          BEGIN
            READ(nffile,nfrec);
            WITH nfrec DO
              IF id = ch THEN
                BEGIN
                  firoot := rtnam;
                  finame := name;
                  found := TRUE
                END {if}
            END; {while}
          IF NOT found THEN
            Center('There is no file with that letter');
          END {else}
        UNTIL (found) OR (exit)
      END; {Addtosearch}

PROCEDURE Checkatim(firoot:rootname; VAR choice:CHAR; VAR exit:BOOLEAN);
  {for choices B and C: check to see if attribute/item files have data}

```

(continued)

```

VAR
  ch : CHAR;
  fnam : stri;
  atrec : Attrrec;
  itrec : Itemrec;
  atfile: Afile;
  itfile: Ifile;
BEGIN
  fnam := CONCAT(volid,firoot,'AT.DTA');
  ASSIGN(atfile,fnam);
  RESET(atfile);
  READ(atfile,atrec);
  IF atrec.atnum = 0 THEN {no attributes exist}
  BEGIN
    clrscr; GOTOXY(1,5);
    Center('Sorry but no attributes have been created for that data base. ');
    WRITELN(Spa(8),'A) Go to the section that creates the attribute table. ');
    WRITELN(Spa(8),'B) Exit to the Main Menu.',CHR(10));
    Center('Type in the LETTER of your choice');
    REPEAT Rdupcase(ch) UNTIL ch IN ['A','B'];
    IF ch = 'A' THEN choice := 'A'
    ELSE exit := TRUE
  END;
  CLOSE(atfile);
  IF (choice = 'C') AND (NOT exit) THEN
  BEGIN
    fnam := CONCAT(volid,firoot,'IM.DTA');
    ASSIGN(itfile,fnam);
    RESET(itfile);
    READ(itfile,itrec);
    IF itrec.inum = 0 THEN {there is no data}
    BEGIN
      clrscr; GOTOXY(1,5);
      Center('Sorry but there is no data in that data base. ');
      WRITELN(Spa(8),'A) Go to the ADD DATA section. ');
      WRITELN(Spa(8),'B) Exit to the Main Menu.',CHR(10));
      Center('Type in the LETTER of your choice');
      REPEAT Rdupcase(ch) UNTIL ch IN ['A','B'];
      IF ch = 'A' THEN choice := 'B'
      ELSE exit := TRUE
    END;
    CLOSE(itfile)
  END {if choice = C}
END; {Checkatim}

PROCEDURE Newatim(firoot:rootname;fname:itemlab;ch:CHAR);
{Create Attribute (AT) and Item (IM) files and initialize first record}
VAR
  tempstr: stri;
  count : INTEGER;
  atrec:attrrec;
  itrec:itemrec;
  atfile: Afile;
  itfile:Ifile;
BEGIN
  WITH atrec DO
  BEGIN
    atnum := 0; atname := fname; qinum := 0;
    FOR count := 0 to maxqualifiers DO
      quals[count] := '';
  END;
  WITH itrec DO
  BEGIN
    fid := ch; iname := fname; inum := 0;
    FOR count := 0 to maxattributes DO
      idata[count] := NUL {lowest value}
    END;
    tempstr := CONCAT(volid,firoot,'AT.DTA');
    ASSIGN(atfile,tempstr);
    REWRITE(atfile);
    WRITE(atfile,atrec);
    CLOSE(atfile);
    tempstr := CONCAT(volid,firoot,'IM.DTA');
    ASSIGN(itfile,tempstr);
    REWRITE(itfile);
    WRITE(itfile,itrec);
  
```



```

CLOSE(itfile);
END; {Newatim}

PROCEDURE Newfile(VAR firoot:rootname; VAR finame:itemlab; VAR exit:BOOLEAN);
    {Get and verify new data base name}
    {Store if not a duplicate else get another name or exit}
VAR
    okay,found : BOOLEAN;
    chx : CHAR;
    len,tempint : INTEGER;
    labstr,tempstr,fnam : stri;
    atfile : Afile;
BEGIN
    gotoxy(1,21);
    Center('Please type in the name of the new data base. ');
    Center('Press RETURN for main menu. ');
    exit := FALSE;
    REPEAT
        okay := TRUE;
        WRITELN;
        WRITE(Spa(20),'==> '); READLN(con,labstr);
        len := LENGTH(labstr);
        IF len = 0 THEN
            exit := TRUE (* EXIT *)
        ELSE IF len > maxalablen THEN
            BEGIN
                okay := FALSE;
                WRITELN(Spa(20),'A name may not exceed ',maxalablen,' characters. ')
            END
        ELSE
            BEGIN
                {Create a file rootname (first 6 non-space characters)}
                tempstr := labstr;
                tempint := POS(' ',tempstr);
                WHILE tempint <> 0 DO
                    BEGIN
                        DELETE(tempstr,tempint,1);
                        tempint := POS(' ',tempstr)
                    END;
                tempint := len - maxroot;
                IF tempint > 0 THEN
                    DELETE(tempstr,maxroot+1,tempint);
                firoot := tempstr;
                Fixa(labstr,finame);
                tempstr := CONCAT(volid,firoot,'AT.DTA');
                ASSIGN(atfile,tempstr);
                {$I-}
                RESET(atfile);
                IF IORESULT = 0 THEN {duplicate name}
                    BEGIN
                        CLOSE(atfile);
                        clrscr;
                        gotoxy(1,5);
                        Center('Sorry, but the first six characters of two data base names ');
                        Center('may not be identical. ');
                        Showbases(chx);
                        gotoxy(1,21);
                        Center('Type in another name please. ');
                        Center('(Press RETURN for main menu. )');
                        OKAY := FALSE
                    END
                ELSE {add name to list of file names}
                    BEGIN
                        RESET(nffile);
                        tempint := 0; found := FALSE;
                        WHILE (NOT EOF(nffile)) AND (NOT found) DO
                            BEGIN
                                READ(nffile,nfrec);
                                IF nfrec.id = '@' THEN found := TRUE;
                                tempint := tempint + 1
                            END;
                        IF found THEN tempint := tempint - 1;
                        WITH nfrec DO
                            BEGIN
                                id := CHR(tempint + 65);

```

(continued)

```

    rtnam := firoot;
    name := fname
END;
SEEK(nffile,tempint);
WRITE(nffile,nfrec);
    Newatim(firoot,fname,nfrec.id)
    END {else add name}
    {$I+}
    END {else create rootname}
UNTIL okay = TRUE
END; {Newfile}

PROCEDURE Removefile(firoot:rootname; fname:itemlab;
                    VAR exit:BOOLEAN; chx:CHAR);
    {Removes a data base (IM and AT files) and replaces
    fileID in Finames.DTA with the '@' character.}
VAR
    tempint : INTEGER;
    found : BOOLEAN;
    ans : stri;
    atfile:Afile;
    itfile:Ifile;
BEGIN
    Center('You are about to permanently remove the data base');
    WRITELN(Spa(20),fname);
    Center('You must type the word YES (in capitals) to do this. ');
    WRITE('==> '); READLN(ans);
    IF ans <> 'YES' THEN exit := TRUE
    ELSE
        BEGIN
            ans := CONCAT(void,firoot,'AT.DTA');
            ASSIGN(atfile,ans);
            ERASE(atfile);
            ans := CONCAT(void,firoot,'IM.DTA');
            ASSIGN(itfile,ans);
            ERASE(itfile);
            RESET(nffile);
            tempint := 0; found := FALSE;
            WHILE (NOT EOF(nffile)) AND (NOT found) DO
                BEGIN
                    READ(nffile,nfrec);
                    IF nfrec.rtnam = firoot THEN found := TRUE;
                    tempint := tempint + 1
                END;
            IF found THEN tempint := tempint - 1;
            WITH nfrec DO
                BEGIN
                    id := '@';
                    rtnam := '';
                    name := '';
                END;
                SEEK(nffile,tempint);
                WRITE(nffile,nfrec);
                exit := TRUE
            END
        END; {Removefile}

PROCEDURE Showtitle(choice:CHAR);
BEGIN
    clrscr; GOTOXY(1,2);
    CASE choice OF
        'A': Center('CREATE A NEW DATA BASE');
        'B': Center('ADD DATA');
        'C': Center('SEARCH THE DATA BASE');
        'D': Center('REMOVE A DATA BASE')
    END {case}
END; {Showtitle}

BEGIN {Getfname}
    clrscr;
    exit := FALSE;
    {$I-}
    tempstr := CONCAT(void, 'FINAMES.DTA');
    ASSIGN(nffile,tempstr);
    RESET(nffile);
    IF (IORESULT <> 0) AND (choice = 'A') THEN {create the name file}

```



```

BEGIN
  REWRITE(nffile);
  WITH nfreq DO      {initialize finame record 0}
  BEGIN
    id := '@';
    rtnam := '';
    name := '';
  END;
  WRITE(nffile,nfreq)
END
ELSE IF IORESULT <> 0 THEN {no data bases available}
BEGIN
  Center('There are no data bases available');
  Center('You will have to create one before doing any further work.');
```

exit := TRUE

```

END
ELSE;      {dummy option}
{$I+}
IF NOT exit THEN
BEGIN
  Showtitle(choice);
  Showbases(chx);
  CASE choice OF
    'A': Newfile(firoot,finame,exit);
    'B'..'D': Addtosearch(firoot,finame,exit,chx)
  END; {case}
  IF NOT exit THEN
    CASE choice OF
      'B','C': Checkatim(firoot,choice,exit);
      'D'      : Removefile(firoot,finame,exit,chx)
    END {case}
  END;
  CLOSE(nffile)
END; {Getfiname}

PROCEDURE Searchbase(firoot:rootname; finame:itemlab); {search data base}
VAR
  exit : BOOLEAN;
  fnam : stri;
  atable : attirtable;
  Trec : Trackrec;
  Trfile : Tfile;

PROCEDURE Wtandrank(VAR It:Ranktable;atn:attrnum;atr:attrdata;
                    qln:qualnum ;wt:INTEGER; wtit:BOOLEAN);
VAR
  J,last,baseval,atrval : INTEGER;

PROCEDURE Ranklist(VAR It:Ranktable; last:INTEGER);
VAR
  J,K,hdx : INTEGER;
  switch : BOOLEAN;
  high : rankitemrec;
BEGIN
  FOR J := 1 TO last-1 DO
  BEGIN
    switch := FALSE;
    high := It[J]; hdx := J;
    FOR K := J + 1 TO last DO
      IF It[K].rating > high.rating THEN
        BEGIN
          switch := TRUE;
          high := It[K];
          hdx := K
        END; {if}
      IF switch THEN {must switch places}
        BEGIN
          It[hdx] := It[J];
          It[J] := high
        END
      END {for J}
    END; {Ranklist}

BEGIN {Wtandrank}
  baseval := ORD(atr);

```

(continued)

```

last := It[0].item.inum; {# of items in array}
FOR J := 1 TO last DO
BEGIN
    atrval := ORD(It[J].item.idata[atn]);
    IF (wtit) AND (atrval > 0) THEN
        It[J].rating := It[J].rating + ROUND(wt*ABS(baseval-atrval)/qln)
    END;
    IF wtit THEN
        Ranklist(It,last)
    END; {Wtandrank}
}

PROCEDURE Searchfor(VAR It:ranktable; atn:attrnum; atr:attrdata;
    fnam:stri; selcode:CHAR; VAR Zflag:flag);
{Get data from file or from ranktable. It[0] used to pass values to output}
VAR
    memsrch : BOOLEAN;
    J,cnt,last : INTEGER;
    itrec : itemrec;
    tempatr : attrdata;
    itfile:ifile;
PROCEDURE Checkrec(VAR cnt:INTEGER);
{look for data in It or on disk}
{changes It and uses selcode,tempatr,atr,memsrch, and j}
VAR
    match : BOOLEAN;
BEGIN
    match := FALSE;
    CASE selcode OF
        'A' : IF tempatr <= atr THEN match := TRUE;
        'B' : IF tempatr = atr THEN match := TRUE;
        'C' : IF tempatr >= atr THEN match := TRUE
    END; {case}
    IF (memsrch) AND (tempatr = NUL) THEN {Include "no data" items}
        match := TRUE;
    IF match THEN
        BEGIN
            cnt := cnt + 1;
            IF memsrch THEN {data is in It}
                It[cnt] := It[J]
            ELSE {data is on disk}
                BEGIN
                    It[cnt].rating := 0; {initialize rating}
                    It[cnt].item := itrec
                END
            END {if match}
        END; {Checkrec}
    }

BEGIN {Searchfor}
    cnt := 0;
    Zflag := 0; {assume no data is present}
    last := It[0].item.inum;
    IF last = 0 THEN {search disk file}
        BEGIN
            memsrch := FALSE;
            ASSIGN(itfile,fnam);
            RESET(itfile);
            READ(itfile,itrec);
            It[0].item.iname := itrec.iname; {data base name}
            WHILE NOT EOF(itfile) DO
                BEGIN
                    READ(itfile,itrec);
                    tempatr := itrec.idata[atn];
                    Checkrec(cnt);
                END;
            CLOSE(itfile);
            IF cnt = 0 THEN Zflag := -1 {No items found in data base}
        END {last=0}
    ELSE {search array It}
        BEGIN
            memsrch := TRUE;
            FOR J := 1 TO last DO
                BEGIN
                    tempatr := IT[J].item.idata[atn];
                    Checkrec(cnt)
                END;
            IF cnt = 0 THEN Zflag := 1 {No items in It fit the new constraint}
        END
    END

```



```

END; {else}
It[0].item.idata[0] := atr;    {desired qualifier value}
IF Zflag <> 1 THEN {change item count. New list or no items in data base}
    IT[0].item.inum := cnt
END; {Searchfor}

PROCEDURE Dumpdata(VAR A :attrtable;VAR IT:ranktable;atn:attrnum;
                   print:markarray;VAR selcode,ldata,dev:CHAR);
    {Output list and rating to console or printer}
VAR
    devnam : stri;
    ch,lf : CHAR;
    outfile : text;

PROCEDURE PAUSE;
BEGIN
    IF DEVNAM = 'con:' THEN
        BEGIN
            GOTOXY(1,24);
            Center('Press any key to continue. ');
            repeat until keypressed;
            clrscr
        END
    END; {PAUSE}

PROCEDURE Outlist; {output list and rating}
VAR
    j,last,pos,loopcount:INTEGER;

PROCEDURE Heading;
BEGIN
    GOTOXY(1,2);
    WRITELN(outfile,'RANKED LISTING':20,'Number of items':20,It[0].item.inum:4);
    WRITELN(outfile,'Data Base: ':18,IT[0].item.iname,lf);
    RESET(trfile);
    WRITELN(outfile,'Attribute':16,'Qualifier ':32,'# of Items':13,lf);
    WHILE NOT EOF(trfile) DO
        BEGIN
            READ(trfile,trec);
            WITH trec DO
                BEGIN
                    WRITE(outfile,atname:23);
                    CASE selection OF
                        'A': WRITE(outfile,' ',A[atnum].quals[1],' to ',qualname);
                        'B': WRITE(outfile,' ',',equal to ',qualname);
                        'C': WRITE(outfile,' ',qualname,' to ',A[atnum].quals[A[atnum].qinum])
                    END; {case}
                    WRITELN(outfile,nofitems:5)
                END {with}
            END; {while}
            WRITELN(outfile,lf,' NO.','ITEM ':20,'RATING':9,lf)
        END; {Heading}

BEGIN {Outlist}
    clrscr;
    Heading;
    last := It[0].item.inum;
    loopcount:= 1;
    FOR J := 1 TO last DO
        BEGIN
            WRITELN(outfile,J:3,' : ',IT[J].item.iname,IT[J].rating:6);
            IF (J MOD 10 = 0) AND (loopcount <> last) AND (devnam = 'con:') THEN
                BEGIN
                    Pause;
                    Heading
                END;
            loopcount := loopcount+1
        END;
        WRITELN(outfile,lf,lf);
        Pause
    END; {Outlist}

PROCEDURE Crtdata;    { Output the actual data to screen }
VAR
    J,K,POS,lasta,lasti,C : INTEGER;

```

(continued)

```

BEGIN
  lasta := A[0].atnum;
  lasti := IT[0].item.inum;
  clrscr; GOTOXY(10,2);
  Writeln('Data Base: ',IT[0].item.iname);
  Write(Spa(19));
  C := 0; {count for linefeeds}
  FOR K := 1 TO lasta DO
    IF PRINT[K] = TRUE THEN
      BEGIN
        Write(A[K].atname);
        C := C + 1;
        IF C MOD 3 = 0 THEN
          BEGIN
            Writeln;
            Write(Spa(19))
          END
        END; {if true}
      Writeln;
      FOR J := 1 TO lasti DO
        BEGIN
          Writeln(Spa(5),IT[J].item.iname);
          C := 0;
          Write(Spa(13));
          FOR K := 1 TO lasta DO
            BEGIN
              POS:= ORD(IT[J].item.ldata[K]);
              IF PRINT[K] = TRUE THEN
                BEGIN
                  Write(A[K].qual[POS]:20);
                  C := C + 1;
                  IF C MOD 3 = 0 THEN
                    BEGIN
                      Writeln;
                    END {if mod}
                END {if true}
              END; {for k}
            Writeln;
            IF J MOD 3 = 0 THEN
              PAUSE
            END;
            IF lasti = 0 THEN
              Writeln(Spa(15),'No items fit this condition.');
```

Writeln

END; {Crtdata}

```

PROCEDURE Printdata; { Output the actual data to printer}
VAR
  J,K,POS,lasta,lasti,C : INTEGER;
BEGIN
  clrscr;
  Writeln(Outfile,lf);
  lasta := A[0].atnum;
  c := 0;
  FOR K := 1 to lasta DO
    IF print[K] = TRUE THEN
      c := c + 1;
    IF c > 3 THEN
      Write(outfile,CHR(15)); {Epson compressed print on}
      Writeln(outfile,'Data Base: ':17,IT[0].item.iname,lf);
      lasti := IT[0].item.inum;
      Write(outfile,'':20);
      C := 0; {count for linefeeds}
      FOR K := 1 TO lasta DO
        IF PRINT[K] = TRUE THEN
          BEGIN
            Write(outfile,A[K].atname);
            C := C + 1;
            IF C MOD 5 = 0 THEN
              Write(outfile,lf,'':20)
            END; {IF TRUE}
          Writeln(outfile);
          FOR J := 1 TO lasti DO
            BEGIN
              Write(outfile,lf,IT[J].item.iname:20);
```



```

C := 0;
FOR K := 1 TO lasta DO
BEGIN
  POS:= ORD(IT[J].item.idata[K]);
  IF PRINT[K] = TRUE THEN
  BEGIN
    WRITE(outfile,A[K].quals[POS]:15,' ');
    C := C + 1;
    IF C MOD 5 = 0 THEN
      WRITE(outfile,lf,'':20)
    END {IF TRUE}
  END; {FOR K}
  WRITELN(outfile)
END; {for j}
IF lasti = 0 THEN
  WRITELN(outfile,'      No items fit this condition. ');
  WRITELN(outfile,lf,CHR(18)) {EPSON COMPRESSED MODE OFF}
END; {Printdata}

PROCEDURE Lmenu(VAR ldata,dev:CHAR);
VAR ans : CHAR;
BEGIN
  GOTOXY(1,22);
  Center('Type the LETTER of your choice. ');
  GOTOXY(1,3);
  Center('DISPLAY DATA FOR ITEMS IN RANKED LIST. ');
  REPEAT
    GOTOXY(1,7);
    WRITELN(Spa(18),'A) List values on the screen. ');
    WRITELN(Spa(18),'B) List values on the printer. ');
    WRITELN(Spa(18),'C) No data list is desired. ');
    Rdupcase(ans)
  UNTIL ans IN ['A'..'C'];
  IF ans = 'C' THEN
    ldata := 'N'
  ELSE
    ldata := 'Y';
  IF ans = 'B' THEN
    dev := 'P'
  ELSE
    dev := 'C'
  END; {Lmenu}

BEGIN {Dumpdata}
  lf := CHR(10); {linefeed character}
  WRITELN(lf);
  IF dev = 'C' THEN
    DEVNAM := 'con:'
  ELSE
    DEVNAM := 'lst: ';
  assign(outfile,devnam);
  RESET(outfile);
  IF LDATA = 'Y' THEN {user wants a listing of the data}
  BEGIN
    IF devnam = 'con:' THEN
      Crtdata
    ELSE
      Printdata;
    PAUSE
  END;
  IF LDATA = 'L' THEN {output ranked list and see if user wants to list data}
  BEGIN
    OUTLIST;
    Lmenu(ldata,dev)
  END;
  CLOSE(outfile)
END; {Dumpdata}

PROCEDURE PICKATTR(VAR A:attrtable; firoot:rootname);
VAR {Main procedure for Searchbase}
  COUNT,WT : INTEGER;
  ch,chx,selcode,dev,lf : CHAR;
  OKAY,SHOW,WTIT : BOOLEAN;
  atn,TEMPatn : attrnum;
  fnam : stri;
  atr : attrdata;

```

(continued)

July

```
ITMATTR : Itemdata;  
Zflag : flag;  
PICKED,PRINT : markarray;  
IT : ranktable;
```

```
PROCEDURE GETQUAL(J:INTEGER;VAR atr:attrdata;VAR selcode:CHAR);  
VAR
```

```
    K : INTEGER;  
    QLN : qualnum;  
    CH,CHX : CHAR;  
BEGIN  
    clrscr;  
    gotoxy(12,5);  
    WRITELN('ATTRIBUTE #',A[J].atnum,' : ',A[J].atname);  
    CHX := '@'; {char that preceeds the letter 'A'}  
    FOR K := 1 TO A[J].qlnum DO  
    BEGIN  
        CHX := SUCC(CHX);  
        WRITELN(Spa(12),CHX:3,' : ',A[J].quals[K]);  
    END;  
    REPEAT  
        gotoxy(1,22);  
        Center('Type the LETTER of the qualifier that will be the');  
        Center('PIVOT (MINIMUM acceptable value) of your search. ');  
        Rdupcase(ch)  
    UNTIL CH IN ['A'..CHX];  
    GOTOXY(1,22);  
    clreol; WRITELN;  
    clreol;  
    QLN := ORD(CH) - 64; {change letter to qualifier #}  
    atr := ITMATTR[QLN]; {qual # to actual qualifier value}  
    REPEAT  
        gotoxy(1,16);  
        Center('Range Selection:');WRITELN;  
        WRITELN(Spa(10),'A: Include qualifiers from "A" to "',ch,'"');  
        WRITELN(Spa(10),'B: Choose qualifier "',ch,'" only');  
        WRITELN(Spa(10),'C: Include qualifiers from "',ch,'" to "',chx,'" if);  
        Center('Type the LETTER of your choice. ');  
        Rdupcase(ch)  
    UNTIL CH IN ['A'..CHX];  
    selcode := CH  
END; {GETQUAL}
```

```
PROCEDURE Showattr(VAR Mark:Markarray;VAR chx:CHAR);  
{Mark is not changed by this proc.}
```

```
VAR  
    J,last : INTEGER;  
BEGIN  
    last := A[0].atnum;  
    clrscr;  
    gotoxy(1,5);  
    Center('CHOOSE ATTRIBUTES');  
    WRITE(If,Spa(8),'CODE ATTRIBUTE');  
    IF last <= 16 THEN WRITELN(If)  
    ELSE  
    BEGIN  
        GOTOXY(40,7);  
        WRITELN('CODE ATTRIBUTE',If)  
    END;  
    CHX := 'A';  
    IF last <= 16 THEN  
        FOR J := 1 TO last DO  
        BEGIN  
            IF MARK[J] THEN  
                WRITELN(Spa(4),'CHOSEN: ',A[J].atname)  
            ELSE  
                WRITELN(Spa(9),CHX,' : ',A[J].atname);  
            CHX := SUCC(CHX)  
        END  
    ELSE  
    BEGIN  
        FOR J := 1 TO 16 DO  
        BEGIN  
            IF MARK[J] THEN  
                WRITELN(Spa(4),'CHOSEN: ',A[J].atname)  
            ELSE
```



```

        WRITELN(Spa(9),CHX,' : ',A[J].atname);
        CHX := SUCC(CHX)
    END;
    FOR J := 17 TO last DO
    BEGIN
        GOTOXY(40,J-8);
        IF MARK[J] THEN
            WRITELN('CHOSEN: ',A[J].atname)
        ELSE
            WRITELN(' ',CHX,' : ',A[J].atname);
            CHX := SUCC(CHX)
        END
    END {ELSE}
END; {SHOWATTR}

PROCEDURE Getcode(VAR ch, chx:CHAR);
                                {chx not changed by this proc}
BEGIN
    REPEAT
        gotoxy(1,22);
        Center('Type a LETTER to choose an item. ');
        Center('Press SPACEBAR when done. ');
        Rdupcase(ch)
    UNTIL CH IN [' ', 'A'..CHX]
END; {Getcode}

PROCEDURE Getweight(VAR wt:INTEGER);
VAR
    ch : CHAR;
BEGIN
    REPEAT
        clrscr;
        gotoxy(1,5);
        Center('Select the RELATIVE IMPORTANCE (weight) you wish to assign');
        Center('to this attribute in the overall selection process. ');
        GOTOXY(1,8);
        WRITELN(Spa(22), 'A: GREATEST');
        WRITELN(Spa(22), 'B: HIGH');
        WRITELN(Spa(22), 'C: MEDIUM');
        WRITELN(Spa(22), 'D: LOW');
        WRITELN(Spa(22), 'E: LEAST');
        gotoxy(1,22);
        Center('Type in the LETTER of your choice. ');
        Rdupcase(ch)
    UNTIL ch IN ['A'..'E'];
    CASE ch OF
        'A':WT := 50; { MULTIPLE OF TEN USED TO PRODUCE WHOLE NUMBER }
        'B':WT := 40; { VALUE IN THE RANKING PROCEDURE }
        'C':WT := 30;
        'D':WT := 20;
        'E':WT := 10
    END {case}
END; {Getweight}

PROCEDURE SHOWORNO(VAR show:BOOLEAN; numofitems:INTEGER; zfl:flag; VAR dev:CHAR);
VAR
    ans: CHAR;
PROCEDURE Showmenu(VAR ch:CHAR);
BEGIN
    GOTOXY(1,22);
    Center('Type the LETTER of your choice. ');
    REPEAT
        GOTOXY(1,10);
        WRITELN(Spa(18), 'A) Print ranked list on screen. ');
        WRITELN(Spa(18), 'B) Print ranked list on printer. ');
        WRITELN(Spa(18), 'C) No listing desired.', lf);
        Rdupcase(ch)
    UNTIL ch IN ['A'..'C']
END; {Showmenu}

BEGIN {Showorno}
    clrscr;
    gotoxy(1,5);
    ans := 'N'; {ASSUME NO-SHOW DATA}
    IF ZFL = 0 THEN

```

(continued)

```

BEGIN
  WRITELN(Spa(19),NUMOFItemS,' Items meet your specifications.',If);
  Center('Choose an option for listing these items.');
```

 Showmenu(ans)

```
END {IF ZFL = 0 }
ELSE IF ZFL = -1 THEN
  BEGIN
    Center('The data base has no items which fit your requirments.');
```

 ans := 'C' {force a "no show"}

```
  END
  ELSE
    BEGIN
      Center('There are no items that fit the new specifications.');
```

 WRITELN;

 Center('Choose an option for the PREVIOUS list.');

 Showmenu(ans)

```
  END; {ELSE}
  IF ANS = 'C' THEN
    show := FALSE
  ELSE
    show := TRUE;
  IF ans = 'B' THEN
    dev := 'P'
  ELSE dev := 'C'
END; {Showorno}

PROCEDURE SELECTERR(VAR OKAY,WTIT : BOOLEAN);
VAR
  CH : CHAR;
BEGIN
  OKAY := FALSE; {ASSUME NO REPITITION OF ITEM}
  clrscr;
  gotoxy(1,5);
  Center('That item has been selected already');
```

 GOTOXY(1,8);

 Center('NOTE: If you reselect the attribute:');

```
  WRITELN(If,Spa(18),'.1) Only qualifiers that are within the range already');
```

 WRITELN(Spa(23),'.selected will be used in the data search.');

```
  WRITELN(If,Spa(18),'.2) The list will retain its original weighting');
```

 WRITELN(Spa(23),'.for this attribute.');

 gotoxy(1,21);

 Center('Type the LETTER "R" to repeat this item selection.');

 Center('Press any other key to make another selection.');

```
  READ(CH);
  IF CH IN ['R','r'] THEN
    BEGIN
      OKAY := TRUE;
      WTIT := FALSE {SAME ATTRIBUTE THEREFORE, DO NOT WEIGHT IT AGAIN}
    END;
    clrscr
  END; {SELECTERR}

PROCEDURE Initatr;
{Initialize Itmattr array and create tempfile for tracking selections}
VAR
  atr : attrdata;
  J : INTEGER;
  tempstr : str1;
BEGIN
  tempstr := CONCAT(volid,'TEMPFILE.DTA');
  ASSIGN(trfile,tempstr);
  REWRITE(trfile);
  atr := NUL;
  ITMATTR[0] := atr;
  FOR J := 1 TO maxqualifiers DO
    BEGIN
      atr := SUCC(atr);
      ITMATTR[J] := atr
    END
  END; {Initatr}

PROCEDURE Initpick(VAR MARK:markarray;VAL:BOOLEAN);
{Init a marker array}
VAR
  J : INTEGER;
```



```

BEGIN
FOR J := 1 TO maxattributes DO
  MARK[J] := VAL
END; {Initpick}

PROCEDURE Keeptrack(numitm:INTEGER; atn:attrnum; atr:attrdata; sel:CHAR);
  {Store the search criteria in a temporary file for later output}
BEGIN
  IF zflag = 1 THEN numitm := 0; {no items met the requirements}
  WHILE NOT EOF(Trfile) DO
    READ(Trfile,trec);
    WITH trec DO
      BEGIN
        nofitems := numitm;
        atnum := atn;
        atrbute := atr;
        selection := sel;
        atname := A[atn].atname;
        qualname := A[atn].quals[ORD(atr)]
      END;
    WRITE(Trfile,trec)
  END; {Keeptrack}

BEGIN {Pickattr}
  If := CHR(10); {linefeed character}
  COUNT := 0;
  IT[0].item.inum := 0; {marks first time through procedure}
  {Searchfor uses this number to determine what data to accept}
  fnam := CONCAT(void,firoot,'IM.DTA');
  INITatr;
  INITPICK(PICKED,FALSE);
  REPEAT
    SHOWATTR(PICKED,CHX);
    GETCODE(CH,CHX);
    IF CH <> ' ' THEN
      BEGIN
        atn := ORD(CH) - 64; {change letter to attribute#}
        OKAY := TRUE; WTIT := TRUE; {default-no repetition, weight choices}
        IF PICKED[atn] THEN
          SELECTERR(OKAY,WTIT);
        IF OKAY THEN
          BEGIN
            PICKED[atn] := TRUE; {marker for selected item}
            IF WTIT THEN GETWEIGHT(WT);
            GETQUAL(atn,atr,selcode);
            SEARCHFOR(IT,atn,atr,fnam,selcode,Zflag);
            SHOWORNO(SHOW,IT[0].item.inum,Zflag,dev); {dev is the output device}
            Keeptrack(IT[0].item.inum,atn,atr,selcode); {'C'=con: 'P'=prn:}
            IF zflag = 0 THEN {a new list has been generated}
              Wtandrank(IT,atn,atr,A[atn].qinum,wt,wtit);
            IF SHOW THEN
              BEGIN
                CH := 'L'; {Show the ranked list}
                DUMPDATA(A,IT,atn,print,selcode,ch,dev);
                IF CH = 'Y' THEN {Dump returned message to list data}
                  BEGIN
                    INITPICK(PRINT,FALSE);
                    PRINT[atn] := TRUE; {Mark attribute as selected}
                    REPEAT
                      clrscr;
                      gotoxy(1,5);
                      WRITELN(Spa(22),'A) List all attributes. ');
                      WRITELN(Spa(22),'B) List selected attributes. ');
                      gotoxy(1,22);
                      Center('Type in the LETTER of your choice. ');
                      Rdupcase(ch)
                    UNTIL CH IN ['A','B'];
                    IF CH = 'A' THEN
                      INITPICK(PRINT,TRUE)
                    ELSE
                      BEGIN
                        REPEAT
                          SHOWATTR(PRINT,CHX);
                          GETCODE(CH,CHX);
                          IF CH <> ' ' THEN

```

(continued)

```

        BEGIN
            TEMPatn := ORD(CH) - 64; { Change letter to attribute#}
            PRINT[TEMPatn] := TRUE;   {Marker for selected item}
        END
        UNTIL CH = ' '
    END; {ELSE}
    CH := 'Y'; { Reset value of ch so dump will list data }
    DUMPDAT(A,IT,atn,print,selcode,ch,dev)
    END; {if ch = 'Y'}
    CH := '@' {Prevent accidental exit from main loop}
    END {if show}
    END {if okay}
    END {if ch <> ' '}
    UNTIL CH = ' ';
    ERASE(Trfile)
END; {PICKATTR}

PROCEDURE Loadatr(VAR A:attrtable; firoot:rootname; VAR exit:BOOLEAN);
    {Load and check attribute file}
VAR
    J : INTEGER;
    fnam : stri;
    atfile : afile;

BEGIN
    clrscr;
    fnam := CONCAT(volid,firoot,'AT.DTA');
    ASSIGN(atfile,fnam);
    RESET(atfile);
    READ(atfile,A[0]);
    FOR J := 1 TO A[0].atnum DO
        read (atfile,a[J]);
    CLOSE(atfile)
END; {Loadatr}

PROCEDURE Introsb;
BEGIN
    clrscr;
    gotoxy (1,5);
    Center('SEARCH DATA BASE');
    writeln;
    Center('The routine is used to search for the items which fit');
    Center('the values or range of values which you select. ');
    gotoxy(1,22);
    Center('Press any key to begin. ');
    repeat until keypressed;
    clrscr
END; {Introsb}

BEGIN {Searchbase}
    Introsb;
    exit := FALSE;
    Loadatr(atable,firoot,exit);
    IF NOT exit THEN
        Pickattr(atable,firoot);
END;

PROCEDURE Addtobase(firoot:rootname; fname:itemlab);
    {Collect and store the data for the materials in}
    {the MSP system.   BTS 11/10/84 rev 4/10/85}
VAR
    fdex : CHAR;
    itdex: INTEGER;
    atable : attrtable;
    itable : itemtable;

PROCEDURE GETDATA(VAR A:attrtable;VAR I:itemtable; ITDEX:INTEGER;var fdex:char);
VAR {GET ITEM NAME AND VALUES OF EACH ATTRIBUTE}
    J,ITMCNT : INTEGER;
    CH : CHAR;
    NAMSTR : itemlab;

PROCEDURE FIXI (LSTR:stri;VAR ILAB:itemlab);
VAR {CHANGE A STRING INTO AN ITEM LABEL}
    J:INTEGER;

```



```

BEGIN
  ILAB := '';
  FOR J := 1 TO maxlablen DO
    ILAB := CONCAT(ILAB, ' ');
  FOR J := 1 TO LENGTH(LSTR) DO
    ILAB[J] := LSTR[J];
END; {FIXI}

```

```

PROCEDURE Placedata(inum, anum: INTEGER; ch: CHAR);
BEGIN {Put qualifier value in the item record's data array}
  WITH I[inum] DO
    CASE ch OF
      'A': Idata[anum] := D1;
      'B': Idata[anum] := D2;
      'C': Idata[anum] := D3;
      'D': Idata[anum] := D4;
      'E': Idata[anum] := D5;
      'F': Idata[anum] := D6;
      'G': Idata[anum] := D7;
      'H': Idata[anum] := D8;
      'I': Idata[anum] := D9;
      'J': Idata[anum] := D10;
      'X': Idata[anum] := NUL
    {ten categories + NUL}
  END
END; {Placedata}

```

```

PROCEDURE Getqual(J, itmcnt: INTEGER);
VAR
  K : INTEGER;
  ch, chx : CHAR;
BEGIN
  clrscr;
  GOTOXY(1, 4);
  WRITELN(Spa(15), 'Attribute #', A[J].atnum, ': ', A[J].atname);
  chx := '@'; {char that is one before the letter 'A'}
  FOR K := 1 TO A[J].qlnum DO
    BEGIN
      chx := SUCC(chx);
      WRITELN(Spa(15), chx:3, ': ', A[J].quals[K]);
    END;
  REPEAT
    GOTOXY(1, 15);
    Center('Type in the LETTER of the qualifier that best describes');
    WRITELN(Spa(20), 'your item's ', A[J].atname);
    Center('Type the letter X for "No Data Available"');
    Rdupcase(ch)
  UNTIL ch IN ['A'..chx, 'X'];
  Placedata(itmcnt, J, ch)
END; {Getqual}

```

```

PROCEDURE VERIFYITEM(VAR ITM: itemrec; ITMCNT: INTEGER);
VAR {VERIFY DATA FOR AN ITEM.}
  CH, CHX : CHAR;
  J, N, last : INTEGER;
BEGIN
  REPEAT
    clrscr;
    gotoxy(1, 4);
    WRITELN(Spa(18), 'VERIFY DATA FOR ', ITM.iname, CHR(10));
    WRITELN(Spa(17), 'CODE      ATTRIBUTE      QUALIFIER', CHR(10));
    CHX := 'A';
    last := A[0].atnum;
    IF last <= 16 THEN
      FOR J := 1 TO last DO
        BEGIN
          N := ORD(ITM.idata[J]);
          WRITELN(Spa(18), CHX, ': ', A[J].atname:20, A[J].quals[N]:14);
          CHX := SUCC(CHX)
        END
      ELSE
        BEGIN
          FOR J := 1 TO 16 DO
            BEGIN
              N := ORD(ITM.idata[J]);
              WRITELN(' ', CHX, ': ', A[J].atname:20, A[J].quals[N]:14);
            END
          END
        END
      REPEAT

```

(continued)

```

    CHX := SUCC(CHX)
END;
FOR J := 17 TO last DO
BEGIN
    N := ORD(ITM.ldata[J]);
    GOTOXY(40,5+J);
    WRITELN(' ',CHX,' : ',A[J].atname:20,A[J].quals[N]:14);
    CHX := SUCC(CHX)
END
END; {ELSE}
CHX := PRED(CHX); {READJUST CODE}
REPEAT
    GOTOXY(1,22);
    Center('Type a LETTER to change an item or SPACEBAR to continue. ');
    Rdupcase(ch)
    UNTIL CH IN [' ', 'A'..CHX];
    last := ORD(CH) - 64; { CHANGE LETTER TO ATTRIBUTE #}
    IF CH <> ' ' THEN
        GETQUAL(last,ITMCNT)
    UNTIL CH = ' ';
END; {VERIFYITEM}

PROCEDURE GETNAME(VAR ILAB:itemlab);
VAR
    ch : CHAR;
    okay: BOOLEAN;
    len : INTEGER;
    itnam : str1;
BEGIN
    clrscr;
    GOTOXY(1,4);
    Center('Please type in the name of the new item. ');
    WRITELN(Spa(20), 'A name may not exceed ',maxalablen,' characters. ');
    REPEAT
        okay := FALSE;
        GOTOXY(5,7);
        READLN(con,itnam);
        LEN := LENGTH(itnam);
        IF LEN > maxalablen THEN
            BEGIN
                GOTOXY(1,22);
                Center('Too many characters in the name')
            END
        ELSE
            BEGIN
                FIXI(itnam,ilab);
                REPEAT
                    GOTOXY(1,22);
                    Center('Press E to change (edit) name. Press C to continue');
                    Rdupcase(ch);
                    GOTOXY(1,21); delline; delline
                UNTIL ch IN ['C','E'];
                IF ch = 'C' THEN
                    okay := TRUE
                END
            UNTIL okay
        END; {GETNAME}

BEGIN {Getdata}
    I[0].iname := A[0].atname; {database name}
    ITMCNT := 0;
    REPEAT
        ITDEX := ITDEX + 1;
        ITMCNT := ITMCNT + 1;
        GETNAME(NAMSTR);
        i[itmcnt].fid := fdex; {file identifier }
        I[ITMCNT].iname := NAMSTR;
        I[ITMCNT].inum := ITDEX;
        clrscr;
        gotoxy(1,4);
        WRITELN(Spa(18),A[0].atname);
        WRITELN(Spa(18),'Get attributes for ',NAMSTR);
        FOR J := 1 TO A[0].atnum DO
            GETQUAL(J,ITMCNT);
            VERIFYITEM(I[ITMCNT],ITMCNT);

```



```

Center('Type "Q" to quit, any other key to enter another item. ');
Rdupcase(ch);
UNTIL ch = 'Q';
I[0].inum := ITDEX; {total number of items now in the list}
END; {GETDATA}

```

```

PROCEDURE Putawayd(VAR I:Itemtable; firoot:rootname);
VAR
    J,fnsh : INTEGER;
    fnam : stri;
    Itfile : Ifile;
    itrec : itemrec;
BEGIN
    fnam := CONCAT(volid,firoot,'IM.DTA');
    ASSIGN(Itfile,fnam);
    RESET(Itfile);
    READ(Itfile,itrec);
    fnsh := I[0].inum - itrec.inum; {# of items to be added to the file}
    IF fnsh > 0 THEN
        BEGIN
            itrec := I[0];
            SEEK(Itfile,0);
            WRITE(Itfile,itrec);
            SEEK(Itfile,FileSize(Itfile)); {go to end of file}
            FOR J := 1 TO fnsh DO
                write(Itfile,I[J]);
            END;
            CLOSE(Itfile)
        END; {Putawayd}
    END;

```

```

PROCEDURE Loadatrx(VAR A:attrtable; firoot:rootname; VAR ITDEX:INTEGER; VAR fdex:char);
VAR
    J : INTEGER;
    fnam : stri;
    atfile : afile;
    itfile : ifile;
    itrec : itemrec;
BEGIN
    clrscr;
    fnam := CONCAT(volid,firoot,'AT.DTA');
    assign (atfile,fnam);
    RESET(atfile);
    read (atfile,a[0]);
    FOR J := 1 TO A[0].atnum DO
        read (atfile,a[J]);
    CLOSE(atfile);
    fnam := CONCAT(volid,firoot,'IM.DTA');
    assign (itfile,fnam);
    RESET(itfile);
    read (itfile,itrec);
    ITDEX := itrec.inum;
    fdex := itrec.fid;
    SEEK(itfile,ITDEX);
    READ(itfile,itrec);
    clrscr; gotoxy(1,4);
    WRITELN(Spa(18),'there are ',itdex,' items in ',a[0].atname);
    IF itdex > 0 THEN
        WRITELN(Spa(18),'The last item in the list is: ',itrec.iname);
    gotoxy(1,21);
    Center('Press any key to continue. ');
    repeat until keypressed;
    CLOSE(itfile)
END; {Loadatrx}

```

```

PROCEDURE Introab;
BEGIN
    clrscr;
    gotoxy (1,8);
    Center('ADD DATA');
    writeLn;
    writeLn;
    Center('This routine is used to add items to the data base. ');
    writeLn;
    WRITELN(Spa(15),'An item's name may not exceed ',maxalablen,' characters. ');

```

(continued)

```

    gotoxy(1,21);
    Center('Press any key to begin.');
```

repeat until keypressed;

```

    clrscr
END; {Introab}

BEGIN {Addtobase}
    Introab;
    Loadatr(atable,froot,ITDEX,fdex);
    Getdata(atable,itable,ITDEX,fdex);
    Putawayd(itable,froot)
END;

PROCEDURE Newbase(froot:rootname;fname:itemlab);
    {Create attribute table for a data base}
    {BTS 10/24/84 rev 4/10/84 10/3/85}

VAR
    account : INTEGER;
    exit : BOOLEAN;
    Atable : Attrtable;

PROCEDURE FIXQ (LSTR:stri;VAR QLAB:quallab);
VAR
    {change a string into a qualifier label}
    J:INTEGER;
BEGIN
    QLAB := '';
    FOR J := 1 TO maxqlablen DO
        QLAB := CONCAT(QLAB,' ');
    FOR J := 1 TO LENGTH(LSTR) DO
        QLAB[J] := LSTR[J]
    END; {FIXQ}

PROCEDURE Blankattr(VAR A:Attrrec);
VAR
    {initialize an attribute record}
    J : INTEGER;
BEGIN
    A.atnum := 0;
    A.atname := '';
    A.qlnum := 0;
    Fixq('No data',Aquals[0]);
    FOR J := 1 TO maxqualifiers DO
        Aquals[J] := ''
    END; {Blankattr}

PROCEDURE GETATTR(VAR A:Attrrec;VAR account:INTEGER;VAR exit: BOOLEAN);
VAR
    {Get an attribute name}
    LEN : INTEGER;
    atrSTR : attrlab;
    LABSTR : stri;
    CH : CHAR;
    okay : BOOLEAN;

BEGIN
    okay := FALSE;    {assume bad input}
    clrscr;
    REPEAT
        GOTOXY(1,5);
        WRITELN(Spa(5),'Please type in a name for attribute #',account,' ');
        READLN(con,labstr);
        len := LENGTH(labstr);
        IF len IN [1..maxalablen] THEN
            BEGIN
                Fixa(labstr,atrstr);
                A.atnum := account;
                A.atname := atrstr;
                okay := TRUE
            END
        ELSE IF len <> 0 THEN
            WRITELN(Spa(10),'A name may not exceed ',maxalablen,' characters.')
        ELSE
            BEGIN
                GOTOXY(1,20);
                WRITELN(Spa(10),'You have not typed in any name for attribute ',account);
                Center('Press the LETTER "Q" to confirm that this is okay.');
```

Center('Press any other key to go back.');

```

                Rdupcase(ch); clrscr;

```



```

    IF ch = 'Q' THEN exit := TRUE
  END
  UNTIL okay OR exit
END; {GETATTR}

PROCEDURE GETQUAL(VAR A:Attrrec; acount:INTEGER);
VAR
  {Get qualifier names for an attribute}
  QCOUNT,LEN : INTEGER;
  QLSTR : qualab;
  LABSTR : stri;
  CH : CHAR;

BEGIN
  LEN := 1; QCOUNT := 0;
  clrscr;
  gotoxy(30,5);
  WRITELN(Spa(5),'Attribute #',acount,' ',A.atname);
  WRITELN;
  WRITELN(Spa(5),'Qualifier Number (The limit is ',maxqualifiers,')');
  WHILE (qcount < maxqualifiers) AND (len <> 0) DO
  BEGIN
    GOTOXY(1,21);
    Center('Press RETURN to exit. ');
    GOTOXY(12,10+qcount);
    WRITE(qcount+1:2,' ');
    READLN(con,labstr);
    LEN := LENGTH(LABSTR);
    IF LEN IN [1..maxqlablen] THEN
    BEGIN
      FIXQ(LABSTR,QLSTR);
      qcount := qcount + 1;
      Aquals[QCOUNT] := QLSTR {Aquals[0] defaults to 'No data'}
    END
    ELSE IF LEN <> 0 THEN
    BEGIN
      GOTOXY(20,20);
      WRITELN(Spa(10),'A name may not exceed ',maxqlablen,' characters. ');
      GOTOXY(1,10+qcount); clreol
    END
    ELSE
    BEGIN
      gotoxy(20,20);
      WRITELN('You have entered ',QCOUNT,' qualifiers. ');
      Center('Press the LETTER "Q" to confirm that you are finished. ');
      Center('Press any other key to add more qualifiers. ');
      Rdupcase(ch);
      GOTOXY(1,20); delline; delline; delline;
      IF CH = 'Q' THEN LEN := 0
      ELSE LEN := 1
    END;
  END; {while}
  A.qinum := qcount {store the # of qualifiers in the record}
END; {Getqual}

PROCEDURE Checkaq(VAR A:attrtable);
VAR
  {show list and allow user to edit it}
  J,K,ANUM,N : INTEGER;
  CH,CHX : CHAR;
  OKAY : BOOLEAN;

PROCEDURE EDITQ(J,N:INTEGER); {EDIT qualifiers}
VAR
  LEN :INTEGER;
  LABSTR : stri;
  QLSTR : qualab;

BEGIN
  REPEAT
    gotoxy(1,21);
    Center('Type in a new qualifier name');
    Center('Press RETURN to continue. ');
    READLN(con,labstr);
    LEN := LENGTH(LABSTR);
    IF LEN <> 0 THEN
      IF LEN IN [1..maxqlablen] THEN
        BEGIN

```

(continued)

```

        FIXQ(LABSTR,QLSTR);
        A[J].quals[N] := QLSTR
    END
ELSE
    WRITELN(Spa(13),'A name may not exceed ',maxqlablen,' characters.')
UNTIL LEN IN [0..maxqlablen]
END; {EDITQ}

PROCEDURE EDITA(J:INTEGER); {EDIT ATTRIBUTE NAME}
VAR
    LEN : INTEGER;
    LABSTR : stri;
    atrSTR : attrlab;

BEGIN
    REPEAT
        gotoxy(1,21);
        Center('Type in a new attribute name');
        Center('Press RETURN to continue.');
```

READLN(con,labstr);

LEN := LENGTH(LABSTR);

IF LEN <> 0 THEN

 IF LEN IN [1..maxalablen] THEN

 BEGIN

 FIXA(LABSTR,atrSTR);

 A[J].atname := atrSTR

 END

 ELSE

 WRITELN(Spa(13),'A name may not exceed ',maxalablen,' characters.')

 UNTIL LEN IN [0..maxalablen]

END; {EDITA}

BEGIN {Checkaq}

 clrscr;

 Center('Check and edit attributes and qualifiers.');

 WRITELN(A[0].atname);

 ANUM := A[0].atnum;

 IF anum >= 1 THEN

 FOR J := 1 TO ANUM DO

 REPEAT

 OKAY := FALSE; clrscr;

 GOTOXY(1,2);

 CHX := '@';

 WRITELN('ATTRIBUTE #',A[J].atnum);

 WRITELN(CHX,' : ',A[J].atname);

 IF A[J].qlnum > 0 THEN

 BEGIN

 FOR K := 1 TO A[J].qlnum DO

 BEGIN

 CHX := SUCC(CHX);

 WRITELN(CHX:3,' : ',A[J].quals[K]);

 END;

 REPEAT

 GOTOXY(1,21);

 Center('Type @ to change the attribute name,');

 Center('a LETTER to change a qualifier name or');

 Center('Press the SPACE BAR to continue. ');

 Rdupcase(ch)

 UNTIL CH IN [' ', '@'..CHX];

 IF CH = '@' THEN

 EDITA(J)

 ELSE IF CH IN ['A'..CHX] THEN

 BEGIN

 N := ORD(CH) - 64; {change char to qualifier #}

 EDITQ(J,N)

 END

 ELSE

 OKAY := TRUE

 END (*if A[J].qlnum*)

 UNTIL OKAY

 END; {Checkaq}

PROCEDURE Putaway(A:attrtable; froot:rootname);

VAR

 J : INTEGER;

 fnam : stri;


```

atfile : atfile;
BEGIN
  clrscr;
  fname := CONCAT(volid,froot,'AT.DTA');
  assign(atfile,fname);
  RESET(atfile);
  FOR J := 0 TO A[0].atnum DO
    write(atfile,a[j]);
  CLOSE(atfile)
END; {Putaway}

PROCEDURE Intronb;
BEGIN
  clrscr;
  gotoxy(1,8);
  Center('CREATE DATA BASE SHELL');
  writeln;
  writeln;
  Center('This routine is used to name:');
  Center('the data base's attributes and their qualifiers');
  gotoxy(1,21);
  Center('Press any key to continue. ');
  REPEAT UNTIL KEYPRESSED;
END; {Intronb}

BEGIN {Newbase}
  Intronb;
  exit := FALSE;
  Atable[0].atname := finame;
  account := 1;
  WHILE (account <= maxattributes) AND (NOT exit) DO
  BEGIN
    Blankattr(Atable[account]);
    Getattr(Atable[account],account,exit);
    IF (account = 1) AND (exit = TRUE) THEN
      account := 0;
    IF (NOT exit) THEN
    BEGIN
      Getqual(Atable[account],account);
      clrscr;
      GOTOXY(1,5);
      WRITELN(Spa(10),'You have finished creating ',account,' attribute(s). ');
      WRITELN(Spa(10),'The latest of which was ',Atable[account].atname);
      WRITELN;
      WRITELN(Spa(10),'A) Create another attribute. ');
      WRITELN(Spa(10),'B) Go on to the next step (edit the entries). ');
      GOTOXY(1,21);
      Center('Type in the LETTER of your choice. ');
      REPEAT Rdupcase(ch) UNTIL ch IN ['A','B'];
      IF ch = 'B' THEN exit := TRUE
      ELSE account := account + 1
    END; {if not exit}
    Atable[0].atnum := account
  END; {while}
  IF account > 0 THEN
  BEGIN
    Checkaq(Atable);
    Putaway(Atable,froot)
  END
END; (*Newbase*)

PROCEDURE Mainmenu (VAR ch:CHAR);
BEGIN
  Clrscr;
  gotoxy(1,5);
  Center('Materials Selection Program');
  GOTOXY(1,8);
  Center('MAIN MENU');
  GOTOXY(1,11);
  WRITELN(Spa(22),'A) Create a new data base');
  WRITELN(Spa(22),'B) Add data to an existing data base');
  WRITELN(Spa(22),'C) Search for selected items');
  WRITELN(Spa(22),'D) Remove a data base');
  WRITELN(Spa(22),'E) System overview');
  WRITELN(Spa(22),'Q) Quit the program');
  gotoxy(1,21);

```

(continued)

```

Center('Type in the LETTER of your choice');
Rdupcase(ch)
END; (*Mainmenu*)

```

```

PROCEDURE Intromain;
BEGIN
  textbackground(7);
  textcolor(1);
  clrscr;
  gotoxy(1,5);
  Center('MATERIALS SELECTION PROGRAM');
  gotoxy(1,9);
  Center('written by');
  Center('Bro. Tom Sawyer, csc and Michael Pecht');
  writeln;
  writeln;
  Center('Mechanical Engineering Department');
  Center('University of Maryland');
  Center('College Park MD 20742');
  Center('(301) 454-8866');
  gotoxy(1,21);
  Center('Press any key to continue. ');
  REPEAT UNTIL KEYPRESSED;
END; {Intromain}

```

```

PROCEDURE Explain; {Give an overview of the system and its use}
BEGIN
  clrscr;
  GOTOXY(1,4);
  Center('SYSTEM OVERVIEW');
  WRITELN;
  Center('TERMS:');
  WRITELN;
  Center('ATTRIBUTE - A physical property or characteristic of the material. ');
  WRITELN;
  Center('QUALIFIER - One of a range of values that describes the property. ');
  WRITELN;
  Center('RELATIVE IMPORTANCE - The weight you wish to give to an attribute ');
  Center('relative to the other attributes your "ideal" material must have. ');
  WRITELN;
  Center('PIVOT - This is the MINIMUM acceptable value for the qualifier. ');
  WRITELN;
  Center('RANGE - Qualifier values around the pivot. ');
  Center('You may choose only the pivot or all values on either side of it. ');
  Center('The farther away a value lies from the pivot, the more weight it ');
  Center('is given in the calculation of overall acceptability (rank). ');
  GOTOXY(1,23);
  Center('Press ANY KEY to return to the main menu. ');
  REPEAT UNTIL Keypressed
END; {Explain}

```

```

BEGIN {MAIN PROGRAM - MSP}
  Intromain;
  valid := 'A: '; {this is the id of the data file volume}
  REPEAT
    exit := FALSE;
    Mainmenu(ch);
    IF ch IN ['A'..'D'] THEN
      Getfname(firoot, fname, ch, exit);
    IF NOT (exit) THEN
      CASE ch OF
        'A': Newbase(firoot, fname);
        'B': Addtobase(firoot, fname);
        'C': Searchbase(firoot, fname);
        'E': Explain
      END
    UNTIL ch = 'Q';
    textbackground(0);
    clrscr
  END.

```

read-me.pas

"A Material Selection Program," by Brother Tom Sawyer and Michael Pecht. July, page 235.

MSP by Brother Tom Sawyer, csc, and Michael Pecht, Mechanical Engineering Department, University of Maryland, College Park, MD 20742. Drive for data files set to A:. You can change the assignment of <volid> in the main block or insert a user input option.

srchfor.pas

"A Material Selection Program," by Brother Tom Sawyer and Michael Pecht. July, page 235.

```

PROCEDURE Searchfor(VAR It:ranktable; atn:attrnum; atr:attrdata;
                    fnam:stri; selcode:CHAR; VAR Zflag:flag);
{Get data from file or from ranktable. It[0] used to pass values to output}
VAR
  memsrch : BOOLEAN;
  J,cnt,last : INTEGER;
  itrec : itemrec;
  tempatr : attrdata;
  itfile:ifile;

PROCEDURE Checkrec(VAR cnt:INTEGER);
{look for data in It or on disk}
{changes It and uses selcode,tempatr,atr,memsrch, and j}
VAR
  match : BOOLEAN;
BEGIN
  match := FALSE;
  CASE selcode OF
    'A' : IF tempatr <= atr THEN match := TRUE;
    'B' : IF tempatr = atr THEN match := TRUE;
    'C' : IF tempatr >= atr THEN match := TRUE;
  END; {case}
  IF (memsrch) AND (tempatr = NUL) THEN {Include "no data" items}
    match := TRUE;
  IF match THEN
    BEGIN
      cnt := cnt + 1;
      IF memsrch THEN      {data is in It}
        It[cnt] := It[J]
      ELSE                  {data is on disk}
        BEGIN
          It[cnt].rating := 0;    {initialize rating}
          It[cnt].item := itrec
        END
    END {if match}
  END; {Checkrec}

BEGIN {Searchfor}
  cnt := 0;
  Zflag := 0;                {assume no data is present}
  last := It[0].item.inum;
  IF last = 0 THEN            {search disk file}
    BEGIN
      memsrch := FALSE;
      ASSIGN(itfile,fnam);
      RESET(itfile);
      READ(itfile,itrec);
      It[0].item.iname := itrec.iname; {data base name}
      WHILE NOT EOF(itfile) DO
        BEGIN
          READ(itfile,itrec);

```

(continued)

```

    tempatr := itrec.idata[atn];
    Checkrec(cnt);
END;
CLOSE(itfile);
IF cnt = 0 THEN Zflag := -1 {No items found in data base}
END {last=0}
ELSE {search array It}
BEGIN
    memsrch := TRUE;
    FOR J := 1 TO last DO
    BEGIN
        tempatr := IT[J].item.idata[atn];
        Checkrec(cnt)
    END;
    IF cnt = 0 THEN Zflag := 1 {No items in It fit the new constraint}
END; {else}
IT[0].item.idata[0] := atr; {desired qualifier value}
IF Zflag <> 1 THEN {change item count. New list or no items in data base}
    IT[0].item.inum := cnt
END; {Searchfor}

```

wtrank.pas

"A Material Selection Program," by Brother Tom Sawyer and Michael Pecht. July, page 235.

```

PROCEDURE Wtandrank(VAR It:Ranktable;atn:attrnum;atr:attrdata;
                    qln:qualnum ;wt:INTEGER; wtit:BOOLEAN);
VAR
    J,last,baseval,atrval : INTEGER;

PROCEDURE Ranklist(VAR It:Ranktable; last:INTEGER);
VAR
    J,K,hdx : INTEGER;
    switch : BOOLEAN;
    high : rankitemrec;
BEGIN
    FOR J := 1 TO last-1 DO
    BEGIN
        switch := FALSE;
        high := It[J]; hdx := J;
        FOR K := J + 1 TO last DO
            IF It[K].rating > high.rating THEN
            BEGIN
                switch := TRUE;
                high := It[K];
                hdx := K
            END; {if}
        IF switch THEN {must switch places}
        BEGIN
            It[hdx] := It[J];
            It[J] := high
        END
    END {for J}
END; {Ranklist}

BEGIN {Wtandrank}
    baseval := ORD(atr);
    last := It[0].item.inum; {# of items in array}
    FOR J := 1 TO last DO
    BEGIN
        atrval := ORD(It[J].item.idata[atn]);
        IF (wtit) AND (atrval > 0) THEN
            It[J].rating := It[J].rating + ROUND(wt*ABS(baseval-atrval)/qln)
        END;
    IF wtit THEN
        Ranklist(It,last)
    END; {Wtandrank}

```


calc.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

EMODULE Calc;

(* A simple spreadsheet program.

Copyright 1986 by Jonathan Amsterdam. All Rights Reserved.

*)

```
FROM Spreadsheet IMPORT setValue, getValue, setFormula, getFormula,
                        maxRow, maxCol, setAutomatic, setManual, recalculate,
                        operationStatus, Status, clearAll;
```

IMPORT Spreadsheet;

```
FROM DisplayHandler IMPORT setCorner, setPrecision, setColWidth,
                        moveCursor, message;
```

IMPORT DisplayHandler;

FROM CommandProc IMPORT command, CommandType, readCommand;

FROM Misc IMPORT fatal;

FROM Terminal IMPORT Beep;

PROCEDURE init;

BEGIN

Spreadsheet.init;

DisplayHandler.init;

END init;

PROCEDURE doLoop;

VAR c:command;

currCellRow, currCellCol:CARDINAL;

BEGIN

currCellRow := 1;

currCellCol := 1;

LOOP

readCommand(c, currCellRow, currCellCol);

CASE c.type OF

CellRef: newCurrCell(currCellRow, currCellCol, c.row, c.col);

SetValue: setValue(currCellRow, currCellCol, c.value);

checkStatus;

SetFormula: setFormula(currCellRow, currCellCol, c.form);

checkStatus;

Left, Right,

Up, Down: IF moveCursor(c.type) THEN

moveCurrCell(currCellRow, currCellCol, c.type);

END;

NewCorner: setCorner(c.row, c.col);

Precision: setPrecision(c.precision);

ColWidth: setColWidth(c.colWidth);

Automatic: setAutomatic;

Manual: setManual;

Recalc: recalculate;

Copy: doCopy(currCellRow, currCellCol, c.row, c.col);

Clear: clearAll;

Quit: EXIT;

ELSE

fatal('doLoop: unknown command type');

END;

END;

END doLoop;

PROCEDURE doCopy(fromRow, fromCol, toRow, toCol:CARDINAL);

VAR v:REAL;

BEGIN

(* check for status? *)

setValue(toRow, toCol, getValue(fromRow, fromCol));

setFormula(toRow, toCol, getFormula(fromRow, fromCol));

END doCopy;

```
PROCEDURE newCurrCell(VAR currCellRow, currCellCol:CARDINAL;
                      row, col:CARDINAL);
```

(continued)

```

BEGIN
  IF (row >= 1) AND (row <= maxRow()) AND (col>=1) AND (col <= maxCol()) THEN
    currCellRow := row;
    currCellCol := col;
  ELSE
    Beep;
  END;
END newCurrCell;

PROCEDURE moveCurrCell(VAR currCellRow, currCellCol: CARDINAL;
  direction: CommandType);
BEGIN
  CASE direction OF
    Up:   IF currCellRow > 1 THEN DEC(currCellRow); ELSE Beep; END;
    Down: IF currCellRow < maxRow() THEN INC(currCellRow); ELSE Beep; END;
    Left: IF currCellCol > 1 THEN DEC(currCellCol); ELSE Beep; END;
    Right: IF currCellCol < maxCol() THEN INC(currCellCol); ELSE Beep; END;
  ELSE
    fatal('moveCurrCell: unknown direction');
  END;
END moveCurrCell;

PROCEDURE checkStatus;
BEGIN
  CASE operationStatus OF
    OK:      (* do nothing *);
    | RangeError: message("Out of range");
  ELSE
    Beep;
  END;
END checkStatus;

BEGIN
  Init;
  doLoop;
END Calc.

```

diskio.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE DiskIO;

(* For low-level I/O to a temporary disk file. Used in the virtual
implementation of the spreadsheet.

The Volume stuff is specific to MacModula-2's implementation of
the file system. You will have to make changes for your system.

Can't mix reading and writing: only reads can occur between a startRead
and an endRead, and similarly for writes.

*)

```

FROM FileSystem IMPORT File, Volume, ReadBytes, WriteBytes, SetPosition,
  VolumeDriveName, OpenVolume, InternalDriveNumber, Create, OpenFile,
  CloseVolume, CloseFile, Delete, fileErrSysEnable, GetFileLength;
FROM MyStorage2 IMPORT ALLOCATE, DEALLOCATE;
FROM Misc IMPORT assert, fatal;
FROM SYSTEM IMPORT TSIZE, ADR, ADDRESS;
FROM StringStuff IMPORT stringLen;

CONST sectorSize = 512;      (* bytes per sector; must be power of 2 *)
  maxCardinalLog2 = 16;      (* equivalent to # of bits in a CARDINAL *)
  sectorSizeLog2 = 9;        (* log base 2 of sector size *)
  maxSectorsPerAddress = 10; (* max sectors in one disk address *)
  fileName = "sstemp";
  bytesPerWord = 2;

TYPE Sector = CARDINAL;
  DiskAddress = POINTER TO daRec;
  daRec = RECORD

```



```

        size:CARDINAL;
        contents:ARRAY[1..maxSectorsPerAddress] OF Sector;
    END;
    SectorList = POINTER TO sectorRec;
    sectorRec = RECORD
        sector:Sector;
        next:SectorList;
    END;

VAR sectorsWritten:ARRAY[1..maxSectorsPerAddress] OF Sector; (* for writing *)
    nSectorsWritten:[0..maxSectorsPerAddress]; (* for writing & rewriting *)
    curDiskAddress:DiskAddress; (* for reading & rewriting *)
    nSectorsRead:[0..maxSectorsPerAddress]; (* for reading *)
    freeSectors: SectorList; (* list of free sectors *)
    posInSector:CARDINAL; (* position in current sector *)
    nextSector:CARDINAL; (* next available sector in file (at end of file) *)
    file: File; (* the scratch file *)
    volume:Volume;
    rewriting:BOOLEAN; (* TRUE if we are rewriting existing sectors *)
    sectorsPerHigh:CARDINAL; (* # of sectors represented by a 1 in the high
        CARDINAL of SetPosition. See goToSector. *)

PROCEDURE init;
VAR i:CARDINAL;
BEGIN
    freeSectors := NIL;
    nextSector := 0;
    posInSector := 0;
    sectorsPerHigh := 1;
    FOR i := 1 TO maxCardinalLog2 - sectorSizeLog2 DO
        sectorsPerHigh := 2 * sectorsPerHigh;
    END;
    makeScratchFile;
END init;

PROCEDURE clear;
VAR temp:SectorList;
BEGIN
    CloseFile(file);
    Delete(volume, fileName);
    CloseVolume(volume);
    WHILE freeSectors <> NIL DO
        temp := freeSectors;
        freeSectors := freeSectors^.next;
        DISPOSE(temp);
    END;
END clear;

PROCEDURE makeScratchFile;
(* Create a new file. This is Mac-specific. Change for
your own system.
With the MacModula-2 file system, you first have to open a volume
(i.e. disk), then create the file, then open it. With most other
file systems, probably a single call to Create will do it. *)
VAR vname:ARRAY[0..26] OF CHAR;
BEGIN
    VolumeDriveName(InternalDriveNumber, vname);
    (* Use the disk in the Mac's internal disk drive *)
    OpenVolume(volume, vname);
    fileErrSysEnable := FALSE; (* Turn off default error handling *)
    Delete(volume, fileName); (* Delete the file, if it exists *)
    fileErrSysEnable := TRUE; (* Turn on error handling *)
    Create(volume, fileName, "????", "????");
    (* The last two args are creator and filetype; they're unimportant. *)
    OpenFile(file, volume, fileName);
END makeScratchFile;

PROCEDURE startWrite;
BEGIN
    nSectorsWritten := 0;
    rewriting := FALSE;
    posInSector := sectorSize;
END startWrite;

```

(continued)

```

PROCEDURE endWrite():DiskAddress;
VAR da:DiskAddress;
    i:CARDINAL;
BEGIN
    assert(nSectorsWritten <> 0, "endWrite: nothing written");
    (* allocate just enough space for the number of sectors written *)
    da := newDiskAddress(nSectorsWritten);
    FOR i := 1 TO nSectorsWritten DO
        da^.contents[i] := sectorsWritten[i];
    END;
    RETURN da;
END endWrite;

```

```

PROCEDURE startRewrite(da:DiskAddress);
BEGIN
    rewriting := TRUE;
    curDiskAddress := da;
    nSectorsWritten := 0;
    posInSector := sectorSize;
END startRewrite;

```

```

PROCEDURE endRewrite():DiskAddress;
VAR newda:DiskAddress;
    i:CARDINAL;
BEGIN
    assert(nSectorsWritten <> 0, "endRewrite: nothing written");
    IF nSectorsWritten = curDiskAddress^.size THEN
        RETURN curDiskAddress;
    ELSE
        newda := newDiskAddress(nSectorsWritten);
        IF nSectorsWritten < curDiskAddress^.size THEN
            FOR i := 1 TO nSectorsWritten DO
                newda^.contents[i] := curDiskAddress^.contents[i];
            END;
            FOR i := nSectorsWritten+1 TO curDiskAddress^.size DO
                freeSector(curDiskAddress^.contents[i]);
            END;
        ELSE
            FOR i := 1 TO curDiskAddress^.size DO
                newda^.contents[i] := curDiskAddress^.contents[i];
            END;
            FOR i := curDiskAddress^.size+1 TO nSectorsWritten DO
                newda^.contents[i] := sectorsWritten[i];
            END;
        END;
        freeDiskAddress(curDiskAddress);
        RETURN newda;
    END;
END endRewrite;

```

```

PROCEDURE startRead(da:DiskAddress);
BEGIN
    curDiskAddress := da;
    nSectorsRead := 0;
    posInSector := sectorSize;
END startRead;

```

```

PROCEDURE endRead;
BEGIN
    (* There is nothing to do when a read is ended, but I leave this procedure
       here for symmetry. *)
END endRead;

```

```

PROCEDURE WriteReal(r:REAL);
BEGIN
    WriteNBytes(ADR(r), TSIZE(REAL)*bytesPerWord);
END WriteReal;

```

```

PROCEDURE ReadReal(VAR r:REAL);
BEGIN
    ReadNBytes(ADR(r), TSIZE(REAL)*bytesPerWord);
END ReadReal;

```

```

PROCEDURE WriteString(VAR s:ARRAY OF CHAR);
(* Assumes strings <= 255 chars *)
VAR len:CARDINAL;

```



```

    clen:ARRAY[0..1] OF CHAR;
BEGIN
    len := stringLen(s);
    assert(len <= 255, "DiskIO.WriteString: too long");
    clen[0] := CHR(len);
    WriteNBytes(ADR(clen), 1);
    WriteNBytes(ADR(s), len);
END WriteString;

PROCEDURE ReadString(VAR s:ARRAY OF CHAR);
VAR len:CARDINAL;
    clen:ARRAY[0..1] OF CHAR;
BEGIN
    ReadNBytes(ADR(clen), 1);
    len := ORD(clen[0]);
    ReadNBytes(ADR(s), len);

    s[len] := 0C;
END ReadString;

PROCEDURE WriteCard(c:CARDINAL);
BEGIN
    WriteNBytes(ADR(c), TSIZE(CARDINAL)*bytesPerWord);
END WriteCard;

PROCEDURE ReadCard(VAR c:CARDINAL);
BEGIN
    ReadNBytes(ADR(c), TSIZE(CARDINAL)*bytesPerWord);
END ReadCard;

PROCEDURE WriteNBytes(a:ADDRESS; count:CARDINAL);
VAR leftInSector:CARDINAL;
BEGIN
    IF count <> 0 THEN
        IF posInSector = sectorSize THEN (* this sector full; get another *)
            newSector;
            assert(posInSector = 0, "WriteNBytes: failed to get new sector");
        END;
        leftInSector := sectorSize - posInSector;
        IF count > leftInSector THEN
            WriteBytes(file, a, leftInSector);
            INC(posInSector, leftInSector);
            INC(a, leftInSector);
            WriteNBytes(a, count-leftInSector);
        ELSE
            WriteBytes(file, a, count);
            INC(posInSector, count);
        END;
    END;
END WriteNBytes;

PROCEDURE ReadNBytes(a:ADDRESS; count:CARDINAL);
VAR leftInSector, actual:CARDINAL;
BEGIN
    IF count <> 0 THEN
        IF posInSector = sectorSize THEN
            INC(nSectorsRead);
            WITH curDiskAddress^ DO
                IF nSectorsRead > size THEN
                    fatal("ReadNBytes: attempt to read too much");
                ELSE
                    goToSector(contents[nSectorsRead]);
                    posInSector := 0;
                END;
            END;
        END;
        leftInSector := sectorSize - posInSector;
        IF count > leftInSector THEN
            ReadBytes(file, a, leftInSector, actual);
            assert(actual = leftInSector, "ReadNBytes: actual <> leftInSector");
            INC(posInSector, leftInSector);
            INC(a, leftInSector);
            ReadNBytes(a, count-leftInSector);
        ELSE
            ReadBytes(file, a, count, actual);
        END;
    END;
END ReadNBytes;

```

(continued)

```

    assert(actual = count, "ReadNBytes: actual <> count");
    INC(posInSector, count);
END;
END;
END ReadNBytes;

PROCEDURE newSector;
BEGIN
    INC(nSectorsWritten);
    posInSector := 0;
    IF rewriting THEN
        IF nSectorsWritten > curDiskAddress^.size THEN
            brandNewSector;
        ELSE
            goToSector(curDiskAddress^.contents[nSectorsWritten]);
        END;
    ELSE
        brandNewSector;
    END;
END newSector;

PROCEDURE brandNewSector;
VAR temp:SectorList;
BEGIN
    IF freeSectors <> NIL THEN
        sectorsWritten[nSectorsWritten] := freeSectors^.sector;
        goToSector(freeSectors^.sector);
        temp := freeSectors;
        freeSectors := freeSectors^.next;
        DISPOSE(temp);
    ELSE
        sectorsWritten[nSectorsWritten] := nextSector;
        goToSector(nextSector);
        INC(nextSector);
    END;
END brandNewSector;

PROCEDURE goToSector(s:Sector);
(* We have to be careful here because the SetPosition function takes two
16-bit CARDINALs, a high and a low; the actual file position is
high*(2^16) + low, which can't be computed directly. So we calculate

(In init) the number of sectors represented by a 1 in the high position,
and work from there. The correctness of the formula depends on the
sector size being a power of 2.
If the place we're going to is beyond the end of the file, the file is
padded appropriately.
*)
VAR hi, lo, curHi, curLo:CARDINAL;
    blanks:ARRAY[1..sectorSize] OF CHAR;
BEGIN
    hi := s DIV sectorsPerHigh;
    lo := (s MOD sectorsPerHigh)*sectorSize;
    GetFileLength(file, curHi, curLo);
    WHILE (curHi < hi) OR ((curHi = hi) AND (curLo < lo)) DO
        (* file too short; pad it with a sector's worth of junk *)
        SetPosition(file, curHi, curLo);
        WriteBytes(file, ADR(blanks), sectorSize);
        GetFileLength(file, curHi, curLo);
    END;
    SetPosition(file, hi, lo);
END goToSector;

PROCEDURE freeSector(s:Sector);
(* Put the sector in sorted order in the list of free sectors. Keeping
the list sorted should improve disk access time, since the head will only
move in one direction (and possibly not at all) when the next sector is
used. *)
VAR sl, news1:SectorList;
BEGIN
    NEW(news1);
    news1^.sector := s;
    IF (freeSectors = NIL) OR (freeSectors^.sector >= s) THEN
        news1^.next := freeSectors;
        freeSectors := news1;
    
```



```

ELSE
  sl := freeSectors;
  WHILE (sl^.next <> NIL) AND (sl^.next^.sector < s) DO
    sl := sl^.next;
  END;
  news1^.next := sl^.next;
  sl^.next := news1;
END;
END freeSector;

PROCEDURE newDiskAddress(nSectors: CARDINAL): DiskAddress;
VAR da: DiskAddress;
BEGIN
  ALLOCATE(da, TSIZE(CARDINAL)+nSectors*TSIZE(Sector));
  da^.size := nSectors;
  RETURN da;
END newDiskAddress;

PROCEDURE freeDiskAddress(VAR da: DiskAddress);
BEGIN
  IF da <> NIL THEN
    DEALLOCATE(da, TSIZE(CARDINAL)+da^.size*TSIZE(Sector));
    da := NIL;
  END;
END freeDiskAddress;

PROCEDURE freeDiskStorage(VAR da: DiskAddress);
VAR i: CARDINAL;
BEGIN
  FOR i := 1 TO da^.size DO
    freeSector(da^.contents[i]);
  END;
  freeDiskAddress(da);
END freeDiskStorage;

PROCEDURE empty(da: DiskAddress): BOOLEAN;
BEGIN
  RETURN da = nullDiskAddress;
END empty;

BEGIN
  nullDiskAddress := NIL;
END DiskIO.

```

readthis

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

Guide to the Source Code for BYTE's Spreadsheet Programming Project

The spreadsheet program consists primarily of 8 modules:

Calc	(.MOD file only)	Top-level module
CellList		Lists of cells for dependency maintenance
CommandProc		Command processor: reads and parses input
DisplayHandler		In charge of the screen display
Evaluator		Parses and evaluates formulas
Formula		Maintains formula data structures
ScreenHandler		Low-level interface to screen
Spreadsheet.DEF		Definition of the spreadsheet abstract data type

Corresponding to the Spreadsheet definition module are four implementations:

Implementation:	Modules used:
Naive	Spreadsheet1
Dependency	Spreadsheet2
Sparse	Spreadsheet3, Cell1
Virtual	Spreadsheet3, Cell2, DiskIO

(continued)

There is a single Cell definition module, used by both Cell implementation modules. To use a particular implementation, compile the necessary modules and rename the compiled code to omit the number. For instance, to use the sparse implementation, compile Spread3.MOD and Cell11.MOD, yielding Spreadsheet3.REL and Cell11.REL (if you are using MacModula-2; the filename extensions may differ for other compilers). Then rename the REL files to Spreadsheet.REL and Cell.REL. Finally, link.

Besides these modules, there are also some support modules:

```
CharStuff
Misc
MyStorage1
MyStorage2
MyTerminal
NumToString
StringStuff
```

The difference between MyStorage1 and MyStorage2 is that the latter returns NIL when an allocation fails for lack of memory. This is very important for the virtual implementation of the spreadsheet. For testing purposes, MyStorage2 will pretend to run out of memory after only a few K bytes have been allocated. You will want to change this if you intend to use the virtual implementation.

A couple of machine-specific places in the code will need to be changed if you are not using a Macintosh. The ScreenHandler module does some strange things to get around a bug in the Macintosh Quickdraw facility. And the DiskIO module uses MacModula-2's FileSystem module, which is tailored to some extent to the Macintosh file system. You may need to make minor changes for your implementation.

spread3.mod

Programming Project: "Build a Spreadsheet Program," by Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Spreadsheet;

```
(* This module is used for both the sparse and virtual implementations.
   It depends on the module Cell (implementation module Cell1 for sparse,
   Cell2 for virtual).
*)
```

```
FROM Formula IMPORT formula, emptyFormula;
IMPORT Formula;
FROM Evaluator IMPORT evaluateFormula;
FROM Misc IMPORT fatal;
FROM DisplayHandler IMPORT displayCell;
FROM CellList IMPORT cellList, cellRow, cellCol, nextCell, initFindDep,
    nextDep, addToCellList, removeFromCellList, nullCellList, empty;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM Cell IMPORT cell, getCell, setCell, doForAllCells, initSheet, clearSheet;
```

```
CONST maxCard = 65535; (* Largest CARDINAL in this implementation *)
```

```
VAR automatic:BOOLEAN;
```

```
PROCEDURE init;
BEGIN
    automatic := TRUE;
    initSheet;
END init;
```

```
PROCEDURE clearAll;
BEGIN
    clearSheet;
END clearAll;
```



```

PROCEDURE setAutomatic;
BEGIN
    automatic := TRUE;
END setAutomatic;

PROCEDURE setManual;
BEGIN
    automatic := FALSE;
END setManual;

PROCEDURE inRange(row, col: CARDINAL): BOOLEAN;
BEGIN
    RETURN (row >= 1) AND (row <= maxRow()) AND
           (col >= 1) AND (col <= maxCol());
END inRange;

PROCEDURE setValue(row, col: CARDINAL; value: REAL);
VAR c: cell;
BEGIN
    IF inRange(row, col) THEN
        getCell(row, col, c);

        c.value := value;
        c.status := OK;
        setCell(row, col, c);
        operationStatus := OK;
        displayCell(row, col);
        IF automatic THEN
            recalc(row, col);
        END;
    ELSE
        operationStatus := RangeError;
    END;
END setValue;

PROCEDURE getValue(row, col: CARDINAL): REAL;
(* Get the value at row, col. *)
VAR c: cell;
BEGIN
    IF NOT inRange(row, col) THEN
        operationStatus := RangeError;
        RETURN 0.0;
    ELSE
        getCell(row, col, c);
        IF c.status <> OK THEN
            operationStatus := c.status;
            RETURN 0.0;
        ELSE
            operationStatus := OK;
            RETURN c.value;
        END;
    END;
END getValue;

PROCEDURE setFormula(row, col: CARDINAL; f: formula);
VAR c: cell;
BEGIN
    IF inRange(row, col) THEN
        freeDependencies(row, col);
        getCell(row, col, c);
        WITH c DO
            form := f;
            value := 0.0;
            status := OK;
            setCell(row, col, c);
            evaluateFormula(form, row, col, value, status);
            setDependencies(row, col);
            displayCell(row, col);
            IF automatic THEN
                recalc(row, col);
            END;
        END;
    END;
END;

```

(continued)

```

    operationStatus := OK;
ELSE
    operationStatus := RangeError;
END;
END setFormula;

```

```

PROCEDURE getFormula(row, col: CARDINAL): formula;
VAR c: cell;
BEGIN
    IF inRange(row, col) THEN
        operationStatus := OK;
        getCell(row, col, c);
        RETURN c.form;
    ELSE
        operationStatus := RangeError;
        RETURN emptyFormula;
    END;
END getFormula;

```

```

PROCEDURE status(row, col: CARDINAL): Status;
VAR c: cell;
BEGIN
    IF inRange(row, col) THEN
        getCell(row, col, c);
        operationStatus := OK;
        RETURN c.status;
    ELSE
        operationStatus := RangeError;
        RETURN RangeError;
    END;
END status;

```

```

PROCEDURE maxRow(): CARDINAL;
BEGIN
    RETURN maxCard;
END maxRow;

```

```

PROCEDURE maxCol(): CARDINAL;
BEGIN
    RETURN maxCard;
END maxCol;

```

```

PROCEDURE recalculate;
BEGIN
    doForAllCells(evalCell);
    operationStatus := OK;
END recalculate;

```

```

PROCEDURE evalCell(row, col: CARDINAL; VAR c: cell);
VAR val: REAL;
    stat: Status;
BEGIN
    WITH c DO
        IF (status <> Empty) AND (status <> SyntaxError)
            AND (NOT Formula.empty(form)) THEN
            evaluateFormula(form, row, col, val, stat);
            IF (stat <> status) OR (val <> value) THEN
                status := stat;
                value := val;
                displayCell(row, col);
            END;
        END;
    END;
END evalCell;

```

```

PROCEDURE recalc(row, col: CARDINAL);
VAR val: REAL;
    stat: Status;
    c: cell;
BEGIN
    getCell(row, col, c);
    WITH c DO

```



```

IF (status <> Empty) AND (status <> SyntaxError)
    AND (NOT Formula.empty(form)) THEN
    evaluateFormula(form, row, col, val, stat);
    IF (stat <> status) OR (val <> value) THEN
        status := stat;
        value := val;
        setCell(row, col, c);
        displayCell(row, col);
        recalcDependents(c);
    END;
ELSE
    recalcDependents(c);
END;
END;
END recalc;

PROCEDURE recalcDependents(VAR c:cell);
VAR cl:cellList;
BEGIN
    cl := c.dependentCells;
    WHILE NOT empty(cl) DO
        recalc(cellRow(cl), cellCol(cl));
        cl := nextCell(cl);
    END;
END recalcDependents;

PROCEDURE setDependencies(row, col:CARDINAL);
VAR r, c:CARDINAL;
    cel:cell;
BEGIN
    getCell(row, col, cel);
    IF cel.status <> SyntaxError THEN
        initFindDep(row, col, cel.form);
        WHILE nextDep(r, c) DO
            IF inRange(r, c) THEN
                getCell(r, c, cel);
                addToCellList(cel.dependentCells, row, col);
                setCell(r, c, cel);
            END;
        END;
    END;
END setDependencies;

PROCEDURE freeDependencies(row, col:CARDINAL);
VAR r, c:CARDINAL;
    cel:cell;
BEGIN
    getCell(row, col, cel);
    IF cel.status <> SyntaxError THEN
        initFindDep(row, col, cel.form);
        WHILE nextDep(r, c) DO
            IF inRange(r, c) THEN
                getCell(r, c, cel);
                removeFromCellList(cel.dependentCells, row, col);
                setCell(r, c, cel);
            END;
        END;
    END;
END freeDependencies;

PROCEDURE clear(row, col:CARDINAL);
BEGIN
    (* not implemented *)
END clear;
BEGIN
END Spreadsheet.

```

evaluat.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE Evaluator;

FROM Spreadsheet IMPORT Status;
FROM Formula IMPORT formula;

EXPORT QUALIFIED evaluateFormula, evaluateString, refexpr;

PROCEDURE evaluateFormula(f:formula; row, col:CARDINAL;
    VAR v:REAL; VAR s:Status);

PROCEDURE evaluateString(str:ARRAY OF CHAR; row, col:CARDINAL;
    VAR v:REAL; VAR s:Status);

PROCEDURE refexpr(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
    VAR v:REAL; VAR s:Status; addBase:CARDINAL);

END Evaluator.

```

displayh.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE DisplayHandler;

FROM CommandProc IMPORT CommandType;

EXPORT QUALIFIED writePrompt, moveCursor, setPrecision, setColWidth,
    displayCell, displayFormula, init, setCorner, message;

PROCEDURE writePrompt(row, col:CARDINAL);

PROCEDURE moveCursor(direction:CommandType):BOOLEAN;

PROCEDURE setPrecision(p:CARDINAL);
(* number of decimal places to be shown *)

PROCEDURE setColWidth(cw:CARDINAL);

PROCEDURE displayCell(row, col:CARDINAL);

PROCEDURE displayFormula(row, col:CARDINAL);

PROCEDURE setCorner(row, col:CARDINAL);

PROCEDURE init;

PROCEDURE message(s:ARRAY OF CHAR);

END DisplayHandler.

```

evaluate.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

IMPLEMENTATION MODULE Evaluator;

(* Evaluator for the spreadsheet.
*)
(* grammar for formulas: ( {} means "zero or more" )
   <expr> ::= <valexpr> | <valexpr> <relop> <valexpr> |

```



```

        IF <expr> , <expr> , <expr>
<valexpr> ::= <term> { <addop> <term> }
<term> ::= <factor> { <mulop> <factor> }
<factor> ::= real | <cellref> | - <factor> | ( <expr> )
<cellref> ::= [ <refexpr> , <refexpr> ]
<refexpr> ::= <addop> real | real
<relop> ::= < | > | = | <> | >= | <=
<addop> ::= + | -
<mulop> ::= * | /

1 = TRUE, 0 = FALSE.
*)

FROM Misc IMPORT fatal, assert;
FROM Spreadsheet IMPORT maxRow, maxCol, Status, status, getValue;
FROM StringStuff IMPORT string40, string160, stringCopy, findChar;
FROM CharStuff IMPORT isDigit, isWhite;
FROM RealConversions IMPORT StrToReal, RealProcResponses, RealConversionRes;
FROM Formula IMPORT formula;
IMPORT Formula;
FROM DisplayHandler IMPORT message;
FROM StringOps IMPORT Concat;

TYPE
    relOpType = (Less, Greater, Equal, LessEqual, GreaterEqual, NotEqual);

VAR curRow, curCol: CARDINAL;

PROCEDURE evaluateFormula(f: formula; row, col: CARDINAL; VAR v: REAL;
    VAR s: Status);
VAR str: string160;
BEGIN
    Formula.toString(f, str);
    evaluateString(str, row, col, v, s);
END evaluateFormula;

PROCEDURE evaluateString(str: ARRAY OF CHAR; row, col: CARDINAL; VAR v: REAL;
    VAR s: Status);
VAR pos: CARDINAL;
BEGIN
    pos := 0;
    curRow := row;
    curCol := col;
    expr(str, pos, v, s, TRUE);
END evaluateString;

(* <expr> ::= <valexpr> <relop> <valexpr> | IF <expr> , <expr> , <expr> *)
PROCEDURE expr(VAR str: ARRAY OF CHAR; VAR pos: CARDINAL;
    VAR v: REAL; VAR s: Status; eval: BOOLEAN);
VAR v1: REAL;
    rop: relOpType;
BEGIN
    IF nextChar(str, pos) THEN
        IF (str[pos] = 'I') AND (str[pos+1] = 'F') THEN
            INC(pos, 2);
            ifexpr(str, pos, v, s, eval);
        ELSE
            valexpr(str, pos, v, s, eval);
            IF s = OK THEN
                IF nextChar(str, pos) THEN
                    relOp(str, pos, rop, s);
                    IF s <> OK THEN (* shouldn't have looked at next char *)
                        s := OK;
                    ELSE
                        valexpr(str, pos, v1, s, eval);
                        IF s = OK THEN
                            IF applyRelOp(rop, v, v1) THEN
                                v := 1.0;
                            ELSE
                                v := 0.0;
                            END IF
                        END IF
                    END IF
                END IF
            END IF
        END IF
    END IF

```

(continued)

```

        END;
    END;
    END;
    END;
    END;
    ELSE
        s := SyntaxError;
        error(str, pos);
    END;
END expr;

(* IF <expr> , <expr> , <expr> *)
PROCEDURE ifexpr(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
                VAR v:REAL; VAR s:Status; eval:BOOLEAN);
(* ifexpr has to eval both branches, even though it knows the
   value of the test, because we do not separate parsing from evaluation.
   It doesn't cause a problem because there are no side-effects. *)
VAR vTrue, vFalse:REAL;
BEGIN
    expr(str, pos, v, s, eval);
    IF s = OK THEN
        IF (NOT nextChar(str, pos)) OR (str[pos] <> ',') THEN
            s := SyntaxError;
            error(str, pos);
        ELSE
            INC(pos);
            expr(str, pos, vTrue, s, v <> 0.0);
            IF s = OK THEN
                IF (NOT nextChar(str, pos)) OR (str[pos] <> ',') THEN
                    s := SyntaxError;
                    error(str, pos);
                ELSE
                    INC(pos);
                    expr(str, pos, vFalse, s, v = 0.0);
                    IF s = OK THEN
                        IF v = 0.0 THEN
                            v := vFalse;
                        ELSE
                            v := vTrue;
                        END;
                    END;
                END;
            END;
        END;
    END;
END ifexpr;

PROCEDURE relOp(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
                VAR rop:relOpType; VAR s:Status);
BEGIN
    IF str[pos] = '=' THEN
        rop := Equal;
        s := OK;
        INC(pos);
    ELSIF str[pos] = '>' THEN
        IF str[pos+1] = '=' THEN
            rop := GreaterEqual;
            INC(pos, 2);
            s := OK;
        ELSE
            rop := Greater;
            s := OK;
            INC(pos);
        END;
    ELSIF str[pos] = '<' THEN
        IF str[pos+1] = '=' THEN
            rop := LessEqual;
            INC(pos, 2);
            s := OK;
        ELSE
            rop := Less;
            INC(pos);
            s := OK;
        END;
    ELSE
        s := SyntaxError;
        error(str, pos);
    END;
END relOp;

```



```

    s := SyntaxError; (* no message; this isn't a real error *)
END;
END relOp;

PROCEDURE applyRelOp(rop:relOpType; v1, v2:REAL):BOOLEAN;
BEGIN
    CASE rop OF
        Equal:      RETURN v1 = v2;
        NotEqual:   RETURN v1 <> v2;
        Less:       RETURN v1 < v2;
        Greater:    RETURN v1 > v2;
        LessEqual:  RETURN v1 <= v2;

        | GreaterEqual: RETURN v1 >= v2;
    ELSE
        fatal('applyBoolOp: unknown op type');
    END;
END applyRelOp;

(* <valexpr> ::= <term> { <addop> <term> } *)
PROCEDURE valexpr(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
    VAR v:REAL; VAR s:Status; eval:BOOLEAN);
VAR v1:REAL;
    op:CHAR;
BEGIN
    term(str, pos, v, s, eval);
    WHILE (s = OK) AND nextChar(str, pos) DO
        IF NOT addOp(str[pos]) THEN
            RETURN;
        END;
        op := str[pos];
        INC(pos);
        term(str, pos, v1, s, eval);
        IF (s = OK) AND eval THEN
            IF op = '+' THEN
                v := v + v1;
            ELSE
                v := v - v1;
            END;
        END;
    END;
END valexpr;

(* <term> ::= <factor> { <mulop> <factor> } *)
PROCEDURE term(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
    VAR v:REAL; VAR s:Status; eval:BOOLEAN);
VAR v1:REAL;
    op:CHAR;
BEGIN
    factor(str, pos, v, s, eval);
    WHILE (s = OK) AND nextChar(str, pos) DO
        IF NOT mulOp(str[pos]) THEN
            RETURN;
        END;
        op := str[pos];
        INC(pos);
        factor(str, pos, v1, s, eval);
        IF (s = OK) AND eval THEN
            IF op = '*' THEN
                v := v * v1;
            ELSIF v1 = 0.0 THEN
                s := DivByZero;
            ELSE
                v := v / v1;
            END;
        END;
    END;
END term;

(* <factor> ::= real | <cellref> | - <factor> | ( <expr> ) *)
PROCEDURE factor(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
    VAR v:REAL; VAR s:Status; eval:BOOLEAN);

```

(continued)

```

BEGIN
  IF NOT nextChar(str, pos) THEN
    s := SyntaxError;
    error(str, pos);
  ELSIF isDigit(str[pos]) THEN
    parseReal(str, pos, v, s);
  ELSE
    INC(pos);
    CASE str[pos-1] OF
      '[': cellRef(str, pos, v, s, eval);
      | '-': factor(str, pos, v, s, eval);
            v := -v;
      | '(': expr(str, pos, v, s, eval);
            IF s = OK THEN
              IF (NOT nextChar(str, pos)) OR (str[pos] <> ')') THEN
                s := SyntaxError;
                error(str, pos);
              ELSE
                INC(pos);
                END;
            END;
    ELSE
      s := SyntaxError;
      error(str, pos);
    END;
  END;
END factor;

(* <cellref> ::= [ <refexpr> , <refexpr> ]
   Opening [ is already read. *)
PROCEDURE cellRef(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
                  VAR v:REAL; VAR s:Status; eval:BOOLEAN);
VAR vRow, vCol:REAL;
    r, c:CARDINAL;
BEGIN
  refexpr(str, pos, vRow, s, curRow);
  IF s = OK THEN
    IF (NOT nextChar(str, pos)) OR (str[pos] <> ',') THEN
      s := SyntaxError;
      error(str, pos);
    ELSE
      INC(pos);
      refexpr(str, pos, vCol, s, curCol);
      IF s = OK THEN
        IF eval THEN
          rangeCheck(vRow, vCol, r, c, s);
        END;
        IF s = OK THEN
          IF eval THEN
            reference(r, c, v, s);
          END;
        END;
        IF s = OK THEN
          IF nextChar(str, pos) AND (str[pos] = ']') THEN
            INC(pos);
          ELSE
            s := SyntaxError;
            error(str, pos);
          END;
        END;
      END;
    END;
  END;
END cellRef;

(* <refexpr> ::= <addop> real | real *)
PROCEDURE refexpr(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL;
                  VAR v:REAL; VAR s:Status; addBase:CARDINAL);
VAR op:CHAR;
BEGIN
  IF NOT nextChar(str, pos) THEN
    s := SyntaxError;
    error(str, pos);
  ELSE
    IF addOp(str[pos]) THEN

```



```

    op := str[pos];
    INC(pos);
ELSE
    op := 0C;
END;
IF NOT nextChar(str, pos) THEN
    s := SyntaxError;
    error(str, pos);
ELSE
    parseReal(str, pos, v, s);
    IF s = OK THEN
        IF op = '+' THEN
            v := FLOAT(addBase) + v;
        ELSIF op = '-' THEN
            v := FLOAT(addBase) - v;
        END;
    END;
END;
END refexpr;

```

```

PROCEDURE addOp(c:CHAR):BOOLEAN;
BEGIN
    RETURN (c = '+') OR (c = '-');
END addOp;
PROCEDURE mulOp(c:CHAR):BOOLEAN;
BEGIN
    RETURN (c = '*') OR (c = '/');

```

```

END mulOp;

```

```

PROCEDURE rangeCheck(vRow, vCol:REAL; VAR r, c:CARDINAL; VAR s:Status);
BEGIN
    IF (vRow >= 1.0) AND (vRow <= FLOAT(maxRow())) AND
       (vCol >= 1.0) AND (vCol <= FLOAT(maxCol())) THEN
        s := OK;
        r := TRUNC(vRow);
        c := TRUNC(vCol);
    ELSE
        s := RangeError;
    END;
END rangeCheck;

```

```

PROCEDURE reference(row, col:CARDINAL; VAR v:REAL; VAR s:Status);
BEGIN
    IF status(row, col) = OK THEN
        v := getValue(row, col);
        s := OK;
    ELSE
        s := RefError;
    END;
END reference;

```

```

PROCEDURE parseReal(VAR str:ARRAY OF CHAR; VAR pos:CARDINAL; VAR v:REAL;
                    VAR s:Status);
VAR real,msg:string40;
    endPos:CARDINAL;
BEGIN
    skipToEndOfReal(str, pos, endPos);
    stringCopy(real, str, pos, endPos);
    StrToReal(real, v);
    CASE RealConversionRes OF
        noError:    s := OK;
    | invalidStr:  s := SyntaxError;
                    Concat(msg, "Invalid real: ", real);
                    message(msg);
    | overflow:    s := Overflow;
    | underflow:   s := Underflow;
    ELSE
        fatal("parseReal: unknown error");
    END;
    pos := endPos+1;
END parseReal;

```

(continued)

```

PROCEDURE skipToEndOfReal(str:ARRAY OF CHAR; pos:CARDINAL;VAR endPos:CARDINAL);
BEGIN
  endPos := pos;
  WHILE (endPos <= HIGH(str)) AND
    findChar("0123456789E.", str[endPos], pos) DO
    INC(endPos);
  END;
  DEC(endPos);
END skipToEndOfReal;

```

mytermin.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

DEFINITION MODULE MyTerminal;

(* Some small but useful additions to the Terminal module. *)

EXPORT QUALIFIED WriteString, WriteLn, Write, Read, ClearScreen, Beep,
WriteLnString, WriteInt, WriteCard, pause, spaces, places;

```

PROCEDURE WriteString(s:ARRAY OF CHAR);
PROCEDURE WriteLn;
PROCEDURE Write(c:CHAR);
PROCEDURE Read(VAR c:CHAR);
PROCEDURE ClearScreen;
PROCEDURE Beep;

```

```

PROCEDURE WriteLnString(s:ARRAY OF CHAR);
PROCEDURE WriteInt(i:INTEGER; spaces:CARDINAL);
PROCEDURE WriteCard(c, spaces:CARDINAL);

```

```

PROCEDURE pause(msg:ARRAY OF CHAR);
(* Prevents the screen from blanking and returning to the Finder until the
   user hits a key. msg is typed out. *)

```

```

PROCEDURE spaces(n:INTEGER);
(* Prints n spaces, if n > 0. *)

```

```

PROCEDURE places(c:CARDINAL):CARDINAL;
(* Returns the number of places it would take to print c. *)

```

END MyTerminal.

spread2.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Spreadsheet;

```

(*)
The DEPENDENCY implementation is an array with dependency information: each
cell knows which other cells depend on its value. In automatic mode, whenever
a value is changed the tree of dependencies is traversed depth-first, and
the display handler is called for each changed value.

```

*)

```

FROM Formula IMPORT formula, emptyFormula;
IMPORT Formula;
FROM Evaluator IMPORT evaluateFormula;
FROM Misc IMPORT fatal;
FROM DisplayHandler IMPORT displayCell;
FROM CellList IMPORT cellList, cellRow, cellCol, nextCell, initFindDep,
  nextDep, addToCellList, removeFromCellList, nullCellList, empty;

```



```

CONST maxR = 32;
      maxC = 60;

TYPE
  cell = RECORD
    value: REAL;
    form: formula;
    status: Status;
    dependentCells: cellList;
  END;

VAR sheet: ARRAY[1..maxR], [1..maxC] OF cell;
    automatic: BOOLEAN;

PROCEDURE Init;
VAR row, col: CARDINAL;
BEGIN
  automatic := TRUE;
  FOR row := 1 TO maxR DO
    FOR col := 1 TO maxC DO
      WITH sheet[row, col] DO
        status := Empty;
        form := emptyFormula;
        dependentCells := nullCellList;
      END;
    END;
  END;
END Init;

PROCEDURE clearAll;
VAR row, col: CARDINAL;
BEGIN
  FOR row := 1 TO maxR DO
    FOR col := 1 TO maxC DO
      WITH sheet[row, col] DO
        status := Empty;
        Formula.free(form);
        form := emptyFormula;
      END;
    END;
  END;
END clearAll;

PROCEDURE setAutomatic;
BEGIN
  automatic := TRUE;
END setAutomatic;

PROCEDURE setManual;
BEGIN
  automatic := FALSE;
END setManual;

PROCEDURE inRange(row, col: CARDINAL): BOOLEAN;
BEGIN
  RETURN (row >= 1) AND (row <= maxR) AND (col >= 1) AND (col <= maxC);
END inRange;

PROCEDURE setValue(row, col: CARDINAL; value: REAL);
BEGIN
  IF inRange(row, col) THEN
    sheet[row, col].value := value;
    sheet[row, col].status := OK;
    operationStatus := OK;
    displayCell(row, col);
    IF automatic THEN
      recalc(row, col);
    END;
  ELSE
    operationStatus := RangeError;
  END;
END setValue;

```

(continued)

```

PROCEDURE getValue(row, col: CARDINAL): REAL;
(* Get the value at row, col. *)
BEGIN
  IF NOT inRange(row, col) THEN
    operationStatus := RangeError;
    RETURN 0.0;
  ELSIF sheet[row, col].status <> OK THEN
    operationStatus := sheet[row, col].status;
    RETURN 0.0;
  ELSE
    operationStatus := OK;
    RETURN sheet[row, col].value;
  END;
END getValue;

PROCEDURE setFormula(row, col: CARDINAL; f: formula);
BEGIN
  IF inRange(row, col) THEN
    freeDependencies(row, col);
    WITH sheet[row, col] DO
      form := f;
      value := 0.0;
      status := OK;
      evaluateFormula(form, row, col, value, status);
      setDependencies(row, col);
      displayCell(row, col);
      IF automatic THEN
        recalc(row, col);
      END;
    END;
    operationStatus := OK;
  ELSE
    operationStatus := RangeError;
  END;
END setFormula;

PROCEDURE getFormula(row, col: CARDINAL): formula;
BEGIN
  IF inRange(row, col) THEN
    operationStatus := OK;
    RETURN sheet[row, col].form;
  ELSE
    operationStatus := RangeError;
    RETURN emptyFormula;
  END;
END getFormula;

PROCEDURE clear(row, col: CARDINAL);
BEGIN
  IF inRange(row, col) THEN
    sheet[row, col].status := Empty;
    sheet[row, col].form := emptyFormula;
    sheet[row, col].dependentCells := nullCellList;
    operationStatus := OK;
  ELSE
    operationStatus := RangeError;
  END;
END clear;

PROCEDURE status(row, col: CARDINAL): Status;
BEGIN
  IF inRange(row, col) THEN
    operationStatus := OK;
    RETURN sheet[row, col].status;
  ELSE
    operationStatus := RangeError;
    RETURN RangeError;
  END;
END status;

PROCEDURE maxRow(): CARDINAL;
BEGIN
  RETURN maxR;
END maxRow;

```



```

PROCEDURE maxCol():CARDINAL;
BEGIN
    RETURN maxC;
END maxCol;

PROCEDURE recalculate;
VAR row, col:CARDINAL;
    val:REAL;
    stat:Status;
BEGIN
    FOR row := 1 TO maxR DO
        FOR col := 1 TO maxC DO
            WITH sheet[row, col] DO
                IF (status <> Empty) AND (status <> SyntaxError)
                    AND (NOT Formula.empty(form)) THEN
                    evaluateFormula(form, row, col, val, stat);
                    IF (stat <> status) OR (val <> value) THEN
                        status := stat;
                        value := val;
                        displayCell(row, col);
                    END;
                END;
            END;
        END;
    END;
    operationStatus := OK;
END recalculate;

PROCEDURE recalc(row, col:CARDINAL);
VAR val:REAL;
    stat:Status;
    cl:cellList;
BEGIN
    WITH sheet[row, col] DO
        IF (status <> Empty) AND (status <> SyntaxError)
            AND (NOT Formula.empty(form)) THEN
            evaluateFormula(form, row, col, val, stat);
            IF (stat <> status) OR (val <> value) THEN
                status := stat;
                value := val;
                displayCell(row, col);
                recalcDependents(sheet[row, col]);
            END;
        ELSE
            recalcDependents(sheet[row, col]);
        END;
    END;
END recalc;

PROCEDURE recalcDependents(VAR c:cell);
VAR cl:cellList;
BEGIN
    cl := c.dependentCells;
    WHILE NOT empty(cl) DO
        recalc(cellRow(cl), cellCol(cl));
        cl := nextCell(cl);
    END;
END recalcDependents;

PROCEDURE setDependencies(row, col:CARDINAL);
VAR r, c:CARDINAL;
BEGIN
    IF sheet[row, col].status <> SyntaxError THEN
        InitFindDep(row, col, sheet[row, col].form);
        WHILE nextDep(r, c) DO
            IF InRange(r, c) THEN
                addToCellList(sheet[r, c].dependentCells, row, col);
            END;
        END;
    END;
END setDependencies;

```

(continued)

```

PROCEDURE freeDependencies(row, col: CARDINAL);
VAR r, c: CARDINAL;
BEGIN
  IF sheet[row, col].status <> SyntaxError THEN
    initFindDep(row, col, sheet[row, col].form);
    WHILE nextDep(r, c) DO
      IF inRange(r, c) THEN
        removeFromCellList(sheet[r, c].dependentCells, row, col);
      END;
    END;
  END;
END freeDependencies;

BEGIN
END Spreadsheet.

```

displayh.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE DisplayHandler;

(* Top row of screen is for display of messages; 2nd, of formulas; 3rd row
is for entry; rest is spreadsheet. *)

```

FROM CommandProc IMPORT CommandType;
FROM MyTerminal IMPORT WriteCard, ClearScreen, Beep;
IMPORT Formula;
FROM Spreadsheet IMPORT getValue, getFormula, inRange, status, Status;
FROM StringStuff IMPORT stringLen, stringAssign, string80, string40;
FROM Misc IMPORT fatal, min;
FROM NumToString IMPORT cardToString, realToString;
FROM ScreenHandler IMPORT setCursorPos, Write, WriteString;
IMPORT ScreenHandler;

```

CONST

```

messageRow = 0;
messageCol = 0;
formulaDisplayRow = 1;
formulaDisplayCol = 0;
promptRow = 2;
promptCol = 0;
colNumberRow = 3;
colNumberCol = 4;
rowNumberRow = 5;
rowNumberCol = 0;
cornerRow = 5;
cornerCol = 4;
initialColWidth = 6;
maxColWidth = 25;
minColWidth = 1;
initialPrecision = 2;
maxPrecision = 6;
minPrecision = 0;
maxScreenRow = 20;
maxScreenCol = 70;

```

```

VAR cellCursorRow, cellCursorCol, cornerCellRow, cornerCellCol,
precision, colWidth, lastRow, lastCol: CARDINAL;

```

PROCEDURE init;

BEGIN

```

  cornerCellRow := 1;
  cornerCellCol := 1;
  cellCursorRow := cornerCellRow;
  cellCursorCol := cornerCellCol;
  precision := initialPrecision;
  colWidth := initialColWidth;
  ScreenHandler.init;
  redisplay;

```

END init;


```

PROCEDURE writePrompt(row, col: CARDINAL);
BEGIN
    clearToEOL(promptRow, promptCol);
    Write([' ');
    WriteCard(row, 0); Write(', '); WriteCard(col, 0);
    WriteString("]: ");
END writePrompt;

```

```

PROCEDURE message(s: ARRAY OF CHAR);
BEGIN
    clearToEOL(messageRow, messageCol);
    WriteString(s);
END message;

```

```

PROCEDURE moveCursor(direction: CommandType): BOOLEAN;
BEGIN
    CASE direction OF
        Up: IF cellCursorRow > cornerCellRow THEN
            DEC(cellCursorRow);
            displayCellCursor;
            RETURN TRUE;
            ELSE
                Beep;
            END;
        | Down: IF cellCursorRow < lastRow THEN
            INC(cellCursorRow);
            displayCellCursor;
            RETURN TRUE;
            ELSE
                Beep;
            END;
        | Left: IF cellCursorCol > cornerCellCol THEN
            DEC(cellCursorCol);
            displayCellCursor;
            RETURN TRUE;
            ELSE
                Beep;
            END;
        | Right: IF cellCursorCol < lastCol THEN
            INC(cellCursorCol);
            displayCellCursor;
            RETURN TRUE;
            ELSE
                Beep;
            END;
        ELSE
            fatal('moveCursor: unknown direction');
        END;
    RETURN FALSE;
END moveCursor;

```

```

PROCEDURE displayCellCursor;
BEGIN
    (* do nothing for now *)
END displayCellCursor;

```

```

PROCEDURE setPrecision(p: CARDINAL);
(* number of decimal places to be shown *)
BEGIN
    IF (p < minPrecision) OR (p > maxPrecision) THEN
        message("Illegal precision");
    ELSE
        precision := p;
        displayCells;
    END;
END setPrecision;

```

```

PROCEDURE setColWidth(cw: CARDINAL);
BEGIN
    IF (cw < minColWidth) OR (cw > maxColWidth) THEN
        message("Illegal column width");
    ELSE
        colWidth := cw+1;
        redisplay;
    END;

```

(continued)

```

END;
END setColWidth;

PROCEDURE setCorner(row, col: CARDINAL);
BEGIN
  IF inRange(row, col) THEN
    cornerCellRow := row;
    cornerCellCol := col;
    cellCursorRow := row;
    cellCursorCol := col;
    redisplay;
  ELSE
    message("Cell out of range");
  END;
END setCorner;

PROCEDURE displayFormula(row, col: CARDINAL);
BEGIN
  clearToEOL(formulaDisplayRow, formulaDisplayCol);
  Formula.write(getFormula(row, col));
END displayFormula;

PROCEDURE displayCell(row, col: CARDINAL);
VAR dummy: BOOLEAN;
BEGIN
  dummy := dispCell(row, col);
END displayCell;

PROCEDURE dispCell(row, col: CARDINAL): BOOLEAN;
(* Returns TRUE if it actually displays the cell *)
VAR screenRow, screenCol: CARDINAL;
BEGIN
  IF (row >= cornerCellRow) AND (col >= cornerCellCol) THEN
    screenRow := (row - cornerCellRow) + cornerRow;
    screenCol := (col - cornerCellCol) * colWidth + cornerCol;
    IF (screenRow <= maxScreenRow) AND (screenCol <= maxScreenCol) THEN
      setCursorPos(screenRow, screenCol);
      Write('|');
      displayValue(row, col, min(maxScreenCol - screenCol + 1, colWidth - 1));
      RETURN TRUE;
    END;
  END;
  RETURN FALSE;
END dispCell;

PROCEDURE displayValue(row, col, space: CARDINAL);
VAR string: string40;
    i, len: CARDINAL;
BEGIN
  IF space > 0 THEN
    valToString(row, col, string);
    len := stringLen(string);
    IF len >= space THEN
      FOR i := 0 TO space - 1 DO
        Write(string[i]);
      END;
    ELSE
      spaces(space - len);
      IF len <> 0 THEN
        FOR i := 0 TO len - 1 DO
          Write(string[i]);
        END;
      END;
    END;
  END;
END displayValue;

PROCEDURE valToString(row, col: CARDINAL; VAR string: ARRAY OF CHAR);
BEGIN
  CASE status(row, col) OF
    OK:      realToString(getValue(row, col), precision, string);
  
```



```

Empty:      string[0] := 0C;
DivByZero:  stringAssign(string, "?DIV");
RefError:   stringAssign(string, "?REF");
RangeError: stringAssign(string, "?RNG");
Overflow:   stringAssign(string, "?OVF");
Underflow:  stringAssign(string, "?UNF");
SyntaxError: stringAssign(string, "?SYN");
ELSE
  stringAssign(string, "???");
END;
END valToString;

```

```

PROCEDURE redisplay;
VAR row, col: CARDINAL;
BEGIN
  ClearScreen;
  displayColNumbers;
  displayRowNumbers;
  displayCells;
END redisplay;

```

```

PROCEDURE displayCells;
VAR row, col: CARDINAL;
BEGIN
  row := cornerCellRow;
  col := cornerCellCol;
  WHILE dispCell(row, col) DO
    REPEAT
      INC(col);
    UNTIL NOT dispCell(row, col);
    lastCol := col-1;
    INC(row);
    col := cornerCellCol;
  END;
  lastRow := row-1;
END displayCells;

```

```

PROCEDURE displayColNumbers;
VAR s: string80;
    screenCol, cellCol, space, i: CARDINAL;
BEGIN
  setCursorPos(colNumberRow, colNumberCol);
  screenCol := colNumberCol;
  cellCol := cornerCellCol;
  WHILE screenCol <= maxScreenCol DO
    Write('|');
    cardToString(cellCol, s);
    centerString(s, colWidth-1);
    space := min(maxScreenCol-screenCol, colWidth-1);
    IF space <> 0 THEN
      FOR i := 0 TO space-1 DO
        carefulWrite(s, i);
      END;
    END;
    INC(screenCol, space+1);
    INC(cellCol);
  END;
  setCursorPos(colNumberRow+1, 0);
  FOR i := 0 TO maxScreenCol DO
    Write('-');
  END;
END displayColNumbers;

```

```

PROCEDURE displayRowNumbers;
VAR s: string40;
    screenRow, cellRow, i, upperLimit: CARDINAL;
BEGIN
  screenRow := rowNumberRow;
  cellRow := cornerCellRow;
  upperLimit := cornerCol-rowNumberCol-2;
  WHILE screenRow <= maxScreenRow DO
    cardToString(cellRow, s);
    setCursorPos(screenRow, rowNumberCol);
    FOR i := 0 TO upperLimit DO
      carefulWrite(s, i);
    END;
  END;
END displayRowNumbers;

```

(continued)

```

        END;
        INC(screenRow);
        INC(cellRow);
    END;
END displayRowNumbers;

    (** utilities **)

PROCEDURE centerString(VAR s:ARRAY OF CHAR; space:CARDINAL);
(* center the string in the given space, padding the left with blanks *)
VAR spaceCount, i, len:CARDINAL;
BEGIN
    len := stringLen(s);
    IF len < space THEN
        spaceCount := (space - len) DIV 2;
        IF spaceCount <> 0 THEN
            FOR i := len TO 0 BY -1 DO
                s[i+spaceCount] := s[i];
            END;
            FOR i := 0 TO spaceCount-1 DO
                s[i] := ' ';
            END;
        END;
    END;
END centerString;

PROCEDURE carefulWrite(VAR s:ARRAY OF CHAR; i:CARDINAL);
BEGIN
    IF i >= stringLen(s) THEN
        Write(' ');
    ELSE
        Write(s[i]);
    END;
END carefulWrite;

PROCEDURE clearToEOL(row, col:CARDINAL);
BEGIN
    setCursorPos(row, col);
    spaces(maxScreenCol - col + 1);
    setCursorPos(row, col);
END clearToEOL;

PROCEDURE spaces(n:CARDINAL);
BEGIN
    WHILE n > 0 DO
        Write(' ');
        DEC(n);
    END;
END spaces;

BEGIN
END DisplayHandler.

```

screenha.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE ScreenHandler;

```

FROM QuickDraw1 IMPORT TextMode, srcOr, GetPen, DrawText, QDPtr, Rect,
    DrawChar, Point, MoveTo, Move, FontInfo, GetFontInfo, EraseRect,
    SetRect, TextWidth, CharWidth;
FROM StringOps IMPORT Length;
FROM SYSTEM IMPORT ADR;

```

```

VAR charWidth, charHeight, charAscent, charDescent:INTEGER;

```

```

PROCEDURE Init;
(* Sets up for current font *)
VAR info:FontInfo;

```



```

BEGIN
  TextMode(srcOr);
  GetFontInfo(info);
  WITH info DO
    charWidth := widMax;
    charHeight := ascent + descent + leading;
    charAscent := ascent;
    charDescent := descent;
  END;
END Init;

PROCEDURE setCursorPos(row, col: CARDINAL); (* zero-based *)
BEGIN
  MoveTo(VAL(INTEGER, col)*charWidth, VAL(INTEGER, (row+1))*charHeight);
END setCursorPos;

PROCEDURE getCursorPos(VAR row, col: CARDINAL);
VAR p: Point;
BEGIN
  GetPen(p);
  row := (p.v DIV charHeight)-1;
  col := p.h DIV charWidth;
END getCursorPos;

PROCEDURE Write(ch: CHAR);
BEGIN
  erase(0, charAscent, charWidth, -charDescent);
  DrawChar(ch);
END Write;

PROCEDURE WriteString(s: ARRAY OF CHAR);
VAR len: INTEGER;
BEGIN
  len := VAL(INTEGER, Length(s));
  erase(0, charAscent, TextWidth(QDPtr(ADR(s))), 0, len, -charDescent);
  DrawText(QDPtr(ADR(s)), 0, len);
END WriteString;

PROCEDURE erase(left, top, right, bottom: INTEGER);
VAR penPos: Point;
    blankRect: Rect;
BEGIN
  GetPen(penPos);
  SetRect(blankRect, penPos.h + left, penPos.v - top,
    penPos.h + right, penPos.v - bottom);
  EraseRect(blankRect);
END erase;

BEGIN
  END ScreenHandler.

```

mytermin.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE MyTerminal;

(* Some small but useful additions to the Terminal module. *)

IMPORT Terminal;

VAR powerOfTen: ARRAY[0..4] OF CARDINAL;

PROCEDURE WriteLnString(s: ARRAY OF CHAR);

```

BEGIN
  Terminal.WriteString(s);
  Terminal.WriteLn;
END WriteLnString;

```

(continued)

```

PROCEDURE WriteInt(i:INTEGER; spaces:CARDINAL);
BEGIN
  IF i < 0 THEN
    IF spaces <> 0 THEN
      writeNum(CARDINAL(-i), spaces-1, TRUE);
    ELSE
      writeNum(CARDINAL(-i), 0, TRUE);
    END;
  ELSE
    writeNum(CARDINAL(i), spaces, FALSE);
  END;
END WriteInt;

PROCEDURE WriteCard(c, spaces:CARDINAL);
BEGIN
  writeNum(c, spaces, FALSE);
END WriteCard;

PROCEDURE writeNum(c, spaces:CARDINAL; neg:BOOLEAN);
VAR p:CARDINAL;
    i:INTEGER;
BEGIN
  p := places(c);
  FOR i := 1 TO INTEGER(spaces) - INTEGER(p) DO
    Terminal.Write(' ');
  END;
  IF neg THEN
    Terminal.Write('-');
  END;
  FOR i := p-1 TO 0 BY -1 DO
    Terminal.Write(CHR((c DIV powerOfTen[i]) + ORD('0')));
    c := c MOD powerOfTen[i];
  END;
END writeNum;

PROCEDURE places(c:CARDINAL):CARDINAL;
(* Returns the number of places c takes to print; i.e. trunc(1+log10(c)). *)
VAR i:CARDINAL;
BEGIN
  FOR i := 4 TO 0 BY -1 DO
    IF (c DIV powerOfTen[i]) > 0 THEN
      RETURN i+1;
    END;
  END;
  RETURN 1;
END places;

PROCEDURE pause(msg:ARRAY OF CHAR);
(* Prevents the screen from blanking and returning to the Finder until the
   user hits a key. msg is typed out. *)
VAR ch:CHAR;
BEGIN
  Terminal.WriteString(msg);
  Terminal.Read(ch);
END pause;

PROCEDURE spaces(n:INTEGER);
VAR i:INTEGER;
BEGIN
  FOR i := 1 TO n DO
    Terminal.Write(' ');
  END;
END spaces;

(***) Copies of Terminal procedures (***)

PROCEDURE WriteString(s:ARRAY OF CHAR);
BEGIN
  Terminal.WriteString(s);
END WriteString;

PROCEDURE WriteLn;
BEGIN

```



```

    Terminal.WriteLine;
END WriteLn;

PROCEDURE Write(c:CHAR);
BEGIN
    Terminal.Write(c);
END Write;

PROCEDURE Read(VAR c:CHAR);
BEGIN
    Terminal.Read(c);
END Read;

PROCEDURE ClearScreen;
BEGIN
    Terminal.ClearScreen;
END ClearScreen;

PROCEDURE Beep;
BEGIN
    Terminal.Beep;
END Beep;

BEGIN
    powerOfTen[0] := 1;
    powerOfTen[1] := 10;
    powerOfTen[2] := 100;
    powerOfTen[3] := 1000;
    powerOfTen[4] := 10000;
END MyTerminal.

```

charstuff.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE CharStuff;

(* Useful functions for manipulating characters. *)

EXPORT QUALIFIED TAB, EOF, getChar, ungetChar, toUpper, toLower, toString,
    isAlphaNum, isLetter, isUpper, isLower, isDigit, isWhite;

CONST    TAB = 11C;
          EOF = 0C;      (* end of file *)

(* getChar and ungetChar use InOut's Read procedure but allow you to push
   a character back on the input. Only one character at a time can be
   ungoten. getChar returns EOF on end of file. *)

PROCEDURE getChar():CHAR;
PROCEDURE ungetChar;

(* These next few are useful for classifying characters. *)

PROCEDURE isAlphaNum(c:CHAR):BOOLEAN;
PROCEDURE isLetter(c:CHAR):BOOLEAN;
PROCEDURE isUpper(c:CHAR):BOOLEAN;
PROCEDURE isLower(c:CHAR):BOOLEAN;
PROCEDURE isDigit(c:CHAR):BOOLEAN;
PROCEDURE isWhite(c:CHAR):BOOLEAN;      (* space, tab or newline (EOL) *)

(* This converts a lower-case character to upper-case, or does nothing if
   the character isn't a lower-case character. The same procedure is
   available in module StringStuff under the name charCap. *)

PROCEDURE toUpper(c:CHAR):CHAR;
PROCEDURE toLower(c:CHAR):CHAR;

```

(continued)

```

PROCEDURE toString(c:CHAR; VAR s:ARRAY OF CHAR);
(* Converts the character c into a string. This procedure won't be needed
   for the new Modula-2 standard, because a single character will be
   compatible with a string. *)

```

```

END CharStuff.

```

```

charstuf.mod

```

```

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

```

IMPLEMENTATION MODULE CharStuff;

```

```

FROM InOut IMPORT Read, EOL, WriteString;

```

```

VAR ch:CHAR;
    ungotten:BOOLEAN;

```

```

PROCEDURE getChar():CHAR;
BEGIN

```

```

    IF ungotten THEN
        ungotten := FALSE;

```

```

    ELSE
        Read(ch);

```

```

    END;
    RETURN ch;

```

```

END getChar;

```

```

PROCEDURE ungetChar;

```

```

BEGIN

```

```

    IF ungotten THEN
        WriteString("ungetChar: can only unget one character at a time");
        HALT;

```

```

    ELSE
        ungotten := TRUE;

```

```

    END;

```

```

END ungetChar;

```

```

PROCEDURE isAlphaNum(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN isLetter(c) OR isDigit(c);

```

```

END isAlphaNum;

```

```

PROCEDURE isLetter(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN isUpper(c) OR isLower(c);

```

```

END isLetter;

```

```

PROCEDURE isUpper(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN (c >= 'A') AND (c <= 'Z');

```

```

END isUpper;

```

```

PROCEDURE isLower(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN (c >= 'a') AND (c <= 'z');

```

```

END isLower;

```

```

PROCEDURE isDigit(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN (c >= '0') AND (c <= '9');

```

```

END isDigit;

```

```

PROCEDURE isWhite(c:CHAR):BOOLEAN;

```

```

BEGIN

```

```

    RETURN (c = ' ') OR (c = TAB) OR (c = EOL);

```

```

END isWhite;

```

```

PROCEDURE toUpper(c:CHAR):CHAR;

```

```

BEGIN

```

```

    IF isLower(c) THEN
        RETURN CAP(c);

```



```

ELSE
    RETURN c;
END;
END toUpper;

PROCEDURE toLower(c:CHAR):CHAR;
BEGIN
    IF isUpper(c) THEN
        RETURN CHR(ORD(c) - ORD('A') + ORD('a'));
    ELSE
        RETURN c;
    END;
END toLower;

PROCEDURE toString(c:CHAR; VAR s:ARRAY OF CHAR);
BEGIN
    s[0] := c;
    s[1] := 0C;
END toString;

BEGIN
    ungotten := FALSE;
END CharStuff.

```

spread.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

DEFINITION MODULE Spreadsheet;

(* Defines the spreadsheet data type. There are a variety of implementations:

The DUMB implementation is just an array. In automatic mode, every time a value is set, it recalculates the whole spreadsheet. It calls the display handler to redraw those cells which have changed.

The DEPENDENCY implementation is an array with dependency information: each cell knows which other cells depend on its value. In automatic mode, whenever a value is changed the tree of dependencies is traversed depth-first, and the display handler is called for each changed value.

The SPARSE DEPENDENCY implementation is like the dependency, but uses a sparse array of blocks of cells instead of a real array.

The VIRTUAL SPARSE DEPENDENCY implementation is like the sparse dependency, but will write blocks of cells to the disk when memory is full, and swap them in as needed.

*)

FROM Formula IMPORT formula;

EXPORT QUALIFIED setValue, getValue, setFormula, getFormula, status,
Status, maxRow, maxCol, setAutomatic, setManual, recalculate,
Init, inRange, operationStatus, clearAll;

TYPE Status = (OK, Empty, DivByZero, Overflow, Underflow,
RefError, RangeError, SyntaxError);

VAR operationStatus: Status;

(* Spreadsheet operations (like setValue, etc.) will set this to OK if
success, otherwise to an error status, usually RangeError. *)

PROCEDURE Init;

PROCEDURE setValue(row, col:CARDINAL; value:REAL);

(* Sets the value of the cell at row, col. If automatic recalculation is
turned on, this will result in a recalc. operationStatus = RangeError
if row, col is out of range. *)

(continued)

```

PROCEDURE getValue(row, col: CARDINAL): REAL;
(* Get the value at row, col. Returns 0.0 and sets operationStatus to
RangeError if out of range. *)

PROCEDURE setFormula(row, col: CARDINAL; f: formula);
(* Sets the formula. Computes the value for the cell and displays it.
operationStatus = RangeError if row, col out of range. *)

PROCEDURE getFormula(row, col: CARDINAL): formula;
(* Returns the formula. operationStatus = RangeError if out of range. *)

PROCEDURE clearAll;
(* Clears all the spreadsheet's cells. *)

PROCEDURE status(row, col: CARDINAL): Status;
(* Returns the cell's status. Can set operationStatus to RangeError. *)

PROCEDURE maxRow(): CARDINAL;
(* Returns the maximum row value for the spreadsheet. *)

PROCEDURE maxCol(): CARDINAL;
(* Returns the maximum column value for the spreadsheet. *)

PROCEDURE setAutomatic;
(* Sets recalc mode to automatic. *)

PROCEDURE setManual;
(* Sets recalc mode to manual. *)

PROCEDURE recalculate;
(* Recalculates the spreadsheet. Any cell which changes is redisplayed. *)

PROCEDURE inRange(row, col: CARDINAL): BOOLEAN;
(* Returns TRUE if row, col are within the bounds of the spreadsheet
(row between 1 and maxRow, column between 1 and maxCol). *)

END Spreadsheet.

```

stringst.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE StringStuff;

EXPORT QUALIFIED string40, string80, string160, string255,
    stringCap, charCap, stringLen, stringCopy, stringEqual,
    deleteChar, findChar, stringAssign, insertChar;

TYPE
    string40 = ARRAY[0..40] OF CHAR;
    string80 = ARRAY[0..80] OF CHAR;
    string160 = ARRAY[0..160] OF CHAR;
    string255 = ARRAY[0..255] OF CHAR;

PROCEDURE charCap(ch: CHAR): CHAR;

PROCEDURE stringCap(VAR s: ARRAY OF CHAR);

PROCEDURE stringLen(VAR s: ARRAY OF CHAR): CARDINAL;

PROCEDURE stringAssign(VAR dest: ARRAY OF CHAR; source: ARRAY OF CHAR);

PROCEDURE stringCopy(VAR dest, source: ARRAY OF CHAR; from, to: CARDINAL);

PROCEDURE stringEqual(s1, s2: ARRAY OF CHAR): BOOLEAN;

PROCEDURE deleteChar(VAR s: ARRAY OF CHAR; pos: CARDINAL);

PROCEDURE insertChar(ch: CHAR; VAR s: ARRAY OF CHAR; pos: CARDINAL);

```



```
PROCEDURE findChar(s:ARRAY OF CHAR; ch:CHAR; VAR pos:CARDINAL):BOOLEAN;
END StringStuff.
```

spread1.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Spreadsheet;

(* Defines the spreadsheet data type. There are a variety of implementations:

The DUMB implementation is just an array. In automatic mode, every time a
value is set, it recalculates the whole spreadsheet. It calls the display
handler to redraw those cells which have changed.

*)

```
FROM Formula IMPORT formula, emptyFormula;
IMPORT Formula;
FROM Evaluator IMPORT evaluateFormula;
FROM Misc IMPORT fatal;
FROM DisplayHandler IMPORT displayCell;
```

```
CONST maxR = 32;
      maxC = 60;
```

```
TYPE cell = RECORD
    value: REAL;
    form: formula;
    status: Status;
END;
```

```
VAR sheet: ARRAY[1..maxR], [1..maxC] OF cell;
    automatic:BOOLEAN;
```

```
PROCEDURE init;
VAR row, col:CARDINAL;
BEGIN
    automatic := TRUE;
    FOR row := 1 TO maxR DO
        FOR col := 1 TO maxC DO
            sheet[row, col].status := Empty;
            sheet[row, col].form := emptyFormula;
        END;
    END;
END init;
```

```
PROCEDURE clearAll;
VAR row, col:CARDINAL;
BEGIN
    FOR row := 1 TO maxR DO
        FOR col := 1 TO maxC DO
            WITH sheet[row, col] DO
                status := Empty;
                Formula.free(form);
                form := emptyFormula;
            END;
        END;
    END;
END clearAll;
```

```
PROCEDURE setAutomatic;
BEGIN
    automatic := TRUE;
END setAutomatic;
```

```
PROCEDURE setManual;
```

(continued)

```

BEGIN
    automatic := FALSE;
END setManual;

PROCEDURE inRange(row, col: CARDINAL): BOOLEAN;
BEGIN
    RETURN (row >= 1) AND (row <= maxR) AND (col >= 1) AND (col <= maxC);
END inRange;

PROCEDURE setValue(row, col: CARDINAL; value: REAL);
BEGIN
    IF inRange(row, col) THEN
        sheet[row, col].value := value;
        sheet[row, col].status := OK;
        operationStatus := OK;
        displayCell(row, col);
        IF automatic THEN
            recalculate;
        END;
    ELSE
        operationStatus := RangeError;
    END;
END setValue;

PROCEDURE getValue(row, col: CARDINAL): REAL;
BEGIN
    IF NOT inRange(row, col) THEN
        operationStatus := RangeError;
        RETURN 0.0;
    ELSIF sheet[row, col].status <> OK THEN
        operationStatus := sheet[row, col].status;
        RETURN 0.0;
    ELSE
        operationStatus := OK;
        RETURN sheet[row, col].value;
    END;
END getValue;

PROCEDURE setFormula(row, col: CARDINAL; f: formula);
BEGIN
    IF inRange(row, col) THEN
        WITH sheet[row, col] DO
            form := f;
            value := 0.0;
            status := OK;
            evaluateFormula(form, row, col, value, status);
            displayCell(row, col);
            IF automatic THEN
                recalculate;
            END;
        END;
        operationStatus := OK;
    ELSE
        operationStatus := RangeError;
    END;
END setFormula;

PROCEDURE getFormula(row, col: CARDINAL): formula;
BEGIN
    IF inRange(row, col) THEN
        operationStatus := OK;
        RETURN sheet[row, col].form;
    ELSE
        operationStatus := RangeError;
        RETURN emptyFormula;
    END;
END getFormula;

PROCEDURE clear(row, col: CARDINAL);
BEGIN
    IF inRange(row, col) THEN
        sheet[row, col].status := Empty;
        sheet[row, col].form := emptyFormula;
        operationStatus := OK;
    ELSE

```



```

        operationStatus := RangeError;
    END;
END clear;

PROCEDURE status(row, col: CARDINAL): Status;
BEGIN
    IF inRange(row, col) THEN
        operationStatus := OK;
        RETURN sheet[row, col].status;
    ELSE
        operationStatus := RangeError;
        RETURN RangeError;
    END;
END status;

PROCEDURE maxRow(): CARDINAL;
BEGIN
    RETURN maxR;
END maxRow;

PROCEDURE maxCol(): CARDINAL;
BEGIN
    RETURN maxC;
END maxCol;

PROCEDURE recalculate;
VAR row, col: CARDINAL;
    val: REAL;
    stat: Status;
BEGIN
    FOR row := 1 TO maxR DO
        FOR col := 1 TO maxC DO
            WITH sheet[row, col] DO
                IF (status <> Empty) AND (status <> SyntaxError)
                    AND (NOT Formula.empty(form)) THEN
                    evaluateFormula(form, row, col, val, stat);
                    IF (stat <> status) OR (val <> value) THEN
                        status := stat;
                        value := val;
                        displayCell(row, col);
                    END;
                END;
            END;
        END;
    END;
    operationStatus := OK;
END recalculate;

BEGIN
END Spreadsheet.

```

numstostr.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

pxEDEFINITION MODULE NumToString;

EXPORT QUALIFIED cardToString, realToString;

PROCEDURE cardToString(c: CARDINAL;
    VAR s: ARRAY OF CHAR);

PROCEDURE realToString(r: REAL; precision:
    CARDINAL; VAR s: ARRAY OF CHAR);

END NumToString.

```

mystor1.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE MyStorage;

FROM SYSTEM IMPORT ADDRESS;

EXPORT QUALIFIED bytesPerWord, ALLOCATE, DEALLOCATE, available;

CONST bytesPerWord = 2;

PROCEDURE ALLOCATE(VAR a:ADDRESS; nBytes:CARDINAL);

PROCEDURE DEALLOCATE(VAR a:ADDRESS; nBytes:CARDINAL);

PROCEDURE available(nBytes:CARDINAL):BOOLEAN;

END MyStorage.

```

numtostr.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

IMPLEMENTATION MODULE NumToString;

FROM StringStuff IMPORT stringLen, findChar, insertChar, deleteChar;
FROM NumConversions IMPORT CardToStr;
FROM RealConversions IMPORT RealToFormStr, RealProcResponses,
                        RealConversionRes, RealToSciStr;

CONST maxDigits = 7; (* if the number would be larger than this, we use
                        scientific notation and preserve this many digits *)

PROCEDURE cardToString(c:CARDINAL; VAR s:ARRAY OF CHAR);
VAR fieldWidth:CARDINAL;
BEGIN
    (* keep trying to convert until success *)
    fieldWidth := 0;
    REPEAT
        INC(fieldWidth);
        CardToStr(c, s, fieldWidth);
    UNTIL (s[0] <> 0C) (* success *)
        OR (fieldWidth = 5); (* longest cardinal is 5 places long *)
    END cardToString;

PROCEDURE realToString(r:REAL; precision:CARDINAL; VAR s:ARRAY OF CHAR);
BEGIN
    RealToFormStr(r, s, maxDigits+2, precision); (* +2 for sign and dec. pt. *)
    IF RealConversionRes = fieldError THEN
        (* string too long--use scientific notation *)
        RealToSciStr(r, s, 10, 2); (* 10 spaces overall, 2 sig. digits *)
    END;
    cosmeticize(s);
END realToString;

PROCEDURE cosmeticize(VAR s:ARRAY OF CHAR);
(* For reals only. Things to fix:
1. no leading blanks
2. no leading zero: add one
3. trailing zeros: delete them
4. trailing decimal point: remove it
*)
VAR i:CARDINAL;

```



```

BEGIN
  (* delete leading blanks *)
  WHILE s[0] = ' ' DO
    deleteChar(s, 0);
  END;
  (*insert a leading 0 if necessary *)
  IF (s[0] = '.') THEN
    insertChar('0', s, 0);
  END;
  (* remove trailing zeros *)
  i := stringLen(s)-1;

  WHILE s[i] = '0' DO
    s[i] := 0C;
    DEC(i);
  END;
  (* remove trailing decimal point *)
  IF s[i] = '.' THEN
    s[i] := 0C;
  END;
END cosmeticize;

BEGIN
END NumToString.

```

screenha.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE ScreenHandler;

EXPORT QUALIFIED init, setCursorPos, getCursorPos, Write, WriteString;

PROCEDURE init;

PROCEDURE setCursorPos(row, col:CARDINAL);

PROCEDURE getCursorPos(VAR row, col:CARDINAL);

PROCEDURE Write(ch:CHAR);

PROCEDURE WriteString(s:ARRAY OF CHAR);

END ScreenHandler.

```

mystor1.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

IMPLEMENTATION MODULE MyStorage;

(* The "available" function should figure out how much memory is actually
   available, and return TRUE if that is >= its argument. But for the
   purposes of testing, my function just assumes there are 5K available
   and decrements a counter each time it is called. You will want
   to change this if you intend to use the spreadsheet. *)
IMPORT Storage;
FROM SYSTEM IMPORT ADDRESS;

PROCEDURE ALLOCATE(VAR a:ADDRESS; nBytes:CARDINAL);
BEGIN
  IF available(nBytes) THEN
    Storage.ALLOCATE(a, (nBytes+1) DIV bytesPerWord);
  ELSE
    a := NIL;
  END;
END ALLOCATE;

```

(continued)

July

```
PROCEDURE DEALLOCATE(VAR a:ADDRESS; nBytes:CARDINAL);
BEGIN
    Storage.DEALLOCATE(a, (nBytes+1) DIV bytesPerWord);
    a := NIL;
END DEALLOCATE;

VAR freeBytes:CARDINAL;

PROCEDURE available(nBytes:CARDINAL):BOOLEAN;
BEGIN
    IF nBytes <= freeBytes THEN
        DEC(freeBytes, nBytes);
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END;
END available;

BEGIN
    freeBytes := 5 * 1024;    (* 5K for testing purposes *)
END MyStorage.
```

mystir2.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```
DEFINITION MODULE MyStorage2;

FROM SYSTEM IMPORT ADDRESS;

EXPORT QUALIFIED ALLOCATE, DEALLOCATE, available;

PROCEDURE ALLOCATE(VAR a:ADDRESS; nWords:CARDINAL);
PROCEDURE DEALLOCATE(VAR a:ADDRESS; nWords:CARDINAL);
PROCEDURE available(nWords:CARDINAL):BOOLEAN;

END MyStorage2.
```

mystor2.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```
IMPLEMENTATION MODULE MyStorage2;

(* The "available" function should figure out how much memory is actually
   available, and return TRUE if that is >= its argument. But for the
   purposes of testing, my function just assumes there are 6K available
   and decrements a counter each time it is called. You will want
   to change this if you intend to use the spreadsheet. *)

IMPORT Storage;
FROM SYSTEM IMPORT ADDRESS;

VAR freeWords:CARDINAL;

PROCEDURE ALLOCATE(VAR a:ADDRESS; nWords:CARDINAL);
BEGIN
    IF available(nWords) THEN
        Storage.ALLOCATE(a, nWords);
        DEC(freeWords, nWords);
    ELSE
        a := NIL;
    END;
END ALLOCATE;

PROCEDURE DEALLOCATE(VAR a:ADDRESS; nWords:CARDINAL);
```



```

BEGIN
  Storage.DEALLOCATE(a, nWords);
  INC(freeWords, nWords);
  a := NIL;
END DEALLOCATE;

PROCEDURE available(nWords:CARDINAL):BOOLEAN;
BEGIN
  RETURN nWords <= freeWords;
END available;

BEGIN
  freeWords := 3 * 1024;  (* 3K words (6K bytes) for testing purposes *)
END MyStorage2.

```

diskio.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE DiskIO;

(* For low-level I/O to a temporary disk file.  Used in the virtual
   implementation of the spreadsheet. *)

EXPORT QUALIFIED Init, WriteReal, ReadReal, WriteString, ReadString,
  DiskAddress, WriteCard, ReadCard, startRead, startWrite, endRead, empty,
  endWrite, startRewrite, endRewrite, freeDiskStorage, nullDiskAddress,
  clear, freeDiskAddress;

TYPE DiskAddress;

VAR nullDiskAddress:DiskAddress;

PROCEDURE Init;
PROCEDURE clear;
PROCEDURE startWrite;
PROCEDURE endWrite():DiskAddress;
PROCEDURE startRewrite(da:DiskAddress);
PROCEDURE endRewrite():DiskAddress;
PROCEDURE startRead(da:DiskAddress);
PROCEDURE endRead;
PROCEDURE WriteReal(r:REAL);
PROCEDURE ReadReal(VAR r:REAL);
PROCEDURE WriteString(VAR s:ARRAY OF CHAR);
PROCEDURE ReadString(VAR s:ARRAY OF CHAR);
PROCEDURE WriteCard(c:CARDINAL);
PROCEDURE ReadCard(VAR c:CARDINAL);
PROCEDURE freeDiskStorage(VAR da:DiskAddress);
(* Frees the disk address and the associated storage on the disk. *)
PROCEDURE freeDiskAddress(VAR da:DiskAddress);
PROCEDURE empty(da:DiskAddress):BOOLEAN;

END DiskIO.

```

formula.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

DEFINITION MODULE Formula;

EXPORT QUALIFIED formula, toString, toFormula, free, write, empty, length,
emptyFormula;

TYPE formula;

VAR emptyFormula:formula;

PROCEDURE toString(f:formula; VAR s:ARRAY OF CHAR);

PROCEDURE toFormula(s:ARRAY OF CHAR):formula;

PROCEDURE free(VAR f:formula);

PROCEDURE write(f:formula);

PROCEDURE empty(f:formula):BOOLEAN;

PROCEDURE length(f:formula):CARDINAL;

END Formula.

formula.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Formula;

FROM StringStuff IMPORT stringLen, stringAssign;
FROM Misc IMPORT assert;
FROM Terminal IMPORT Write;
FROM MyStorage IMPORT allocate, deallocate;

TYPE formula = POINTER TO ARRAY[0..32000] OF CHAR;

PROCEDURE toString(f:formula; VAR s:ARRAY OF CHAR);
VAR i, len:CARDINAL;

BEGIN

IF f = NIL THEN

s[0] := 0C;

ELSE

len := length(f);

FOR i := 0 TO len-1 DO

IF i > HIGH(s) THEN

RETURN;

ELSE

s[i] := f^[i];

END;

END;

IF len <= HIGH(s) THEN

s[len] := 0C;

END;

END;

END toString;

PROCEDURE toFormula(s:ARRAY OF CHAR):formula;

VAR f:formula;

i, len:CARDINAL;

BEGIN

len := stringLen(s);

IF len = 0 THEN

RETURN NIL;


```

ELSE
  f := new(len);
  FOR i := 0 TO len-1 DO
    f^[i] := s[i];
  END;
  f^[len] := 0C;
  RETURN f;
END;
END toFormula;

PROCEDURE write(f:formula);
VAR i, len:CARDINAL;
BEGIN
  len := length(f);
  IF len <> 0 THEN
    FOR i := 0 TO len-1 DO
      Write(f^[i]);
    END;
  END;
END write;

PROCEDURE empty(f:formula):BOOLEAN;
BEGIN
  RETURN f = NIL;
END empty;

PROCEDURE length(f:formula):CARDINAL;
VAR i:CARDINAL;
BEGIN
  IF f = NIL THEN
    RETURN 0;
  ELSE
    i := 0;
    WHILE f^[i] <> 0C DO
      INC(i);
    END;
    RETURN i;
  END;
END length;

PROCEDURE new(nChars:CARDINAL):formula;
VAR f:formula;
BEGIN
  allocate(f, nChars+1);
  assert(f <> NIL, "Formula.new: out of room");
  RETURN f;
END new;

PROCEDURE free(VAR f:formula);
BEGIN
  IF f <> NIL THEN
    deallocate(f, length(f)+1);
    f := NIL;
  END;
END free;

BEGIN
  emptyFormula := NIL;
END Formula.

```

cellist.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE CellList;

FROM Formula IMPORT formula;

EXPORT QUALIFIED cellList, nextCell, initFindDep, nextDep, cellRow, cellCol,
  addToCellList, removeFromCellList, nullCellList, empty, freeCellList;

```

(continued)

```

TYPE cellList;
VAR nullCellList:cellList;
PROCEDURE empty(cl:cellList):BOOLEAN;
PROCEDURE nextCell(cl:cellList):cellList;
PROCEDURE cellRow(cl:cellList):CARDINAL;
PROCEDURE cellCol(cl:cellList):CARDINAL;
PROCEDURE initFindDep(curRow, curCol:CARDINAL; f:formula);
PROCEDURE nextDep(VAR row, col:CARDINAL):BOOLEAN;
PROCEDURE addToCellList(VAR cl:cellList; row, col:CARDINAL);
PROCEDURE removeFromCellList(VAR cl:cellList; row, col:CARDINAL);
PROCEDURE freeCellList(VAR cl:cellList);
END CellList.

```

commanddp.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE CommandProc;

```

FROM InOut IMPORT ReadString;
FROM Evaluator IMPORT evaluateString;
IMPORT Formula;
FROM Spreadsheet IMPORT Status;
FROM StringStuff IMPORT stringLen, findChar, stringCopy, string160,
    string80, stringCap, deleteChar;
FROM NumConversions IMPORT StrToCard, NumConversionRes, NumProcResponses;
FROM Terminal IMPORT Beep;
FROM DisplayHandler IMPORT writePrompt, displayFormula, message;
FROM StringOps IMPORT Concat;

```

```

VAR commandString: string160;
    curRow, curCol:CARDINAL;

```

```

PROCEDURE readCommand(VAR c:command; crRow, crCol:CARDINAL);
BEGIN

```

```

    curRow := crRow;
    curCol := crCol;
    displayFormula(curRow, curCol);
    LOOP
        writePrompt(curRow, curCol);
        REPEAT
            ReadString(commandString);
            UNTIL commandString[0] <> 0C;
            stringCap(commandString);
            IF commandParse(commandString, c) THEN
                EXIT;
            END;
        END;
    END;

```

```

END readCommand;

```

```

PROCEDURE commandParse(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;

```

```

VAR ch:CHAR;
BEGIN
    ch := s[0];
    deleteChar(s, 0);
    CASE ch OF

```



```

'A':    c.type := Automatic;
'C':    RETURN copy(s, c);
'D':    c.type := Down;
'F':    RETURN setFormula(s, c);
'K':    c.type := Clear;
'L':    c.type := Left;
'M':    c.type := Manual;
'N':    RETURN newCorner(s, c);
'P':    RETURN precision(s, c);
'Q':    c.type := Quit;
'R':    c.type := Right;
'U':    c.type := Up;
'V':    RETURN setValue(s, c);
'W':    RETURN colWidth(s, c);
'!':    c.type := Recalc;
'[':    RETURN doCellRef(s, c);
ELSE
  Beep;
  s[0] := 0C;
  message("Illegal character");
  RETURN FALSE;
END;
RETURN TRUE;
END commandParse;

PROCEDURE copy(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
BEGIN
  c.type := Copy;
  RETURN cellRef(s, c);
END copy;

PROCEDURE setFormula(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
BEGIN
  c.type := SetFormula;
  c.form := Formula.toFormula(s);
  RETURN TRUE;
END setFormula;

PROCEDURE newCorner(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
BEGIN
  c.type := NewCorner;
  RETURN cellRef(s, c);
END newCorner;

PROCEDURE precision(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
BEGIN
  c.type := Precision;
  RETURN cardFromString(s, c.precision);
END precision;

PROCEDURE setValue(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
VAR stat:Status;
BEGIN
  evaluateString(s, curRow, curCol, c.value, stat);
  c.type := SetValue;
  IF stat <> OK THEN
    cantEvalMsg(s);
  END;
  RETURN stat = OK;
END setValue;

PROCEDURE colWidth(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
BEGIN
  c.type := ColWidth;
  RETURN cardFromString(s, c.colWidth);
END colWidth;

PROCEDURE doCellRef(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
VAR b:BOOLEAN;
BEGIN
  b := cellRef1(s, c);
  c.type := CellRef;
  RETURN b;
END doCellRef;

```

(continued)

```

PROCEDURE cellRef(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
(* Syntax: [ <formula> , <formula> ] *)
VAR leftPos:CARDINAL;
BEGIN
  IF NOT findChar(s, '[', leftPos) THEN
    message("Illegal cell specification--no left bracket");
    RETURN FALSE;
  ELSE
    WHILE s[0] <> '[' DO
      deleteChar(s, 0);
    END;
    deleteChar(s, 0);
    RETURN cellRef1(s, c);
  END;
END cellRef;

PROCEDURE cellRef1(VAR s:ARRAY OF CHAR; VAR c:command):BOOLEAN;
(* Syntax: <formula> , <formula> ] *)
VAR st:Status;
    v:REAL;
    rowForm, colForm: string80;
    commaPos, rightPos:CARDINAL;
BEGIN
  IF (NOT findChar(s, ',', commaPos)) OR
    (commaPos = 0) OR
    (commaPos = stringLen(s)-1) THEN
    message("Illegal cell specification--bad comma");
    RETURN FALSE;
  ELSIF (NOT findChar(s, ']', rightPos)) OR (rightPos < commaPos) THEN
    message("Illegal cell specification--bad right bracket");
    RETURN FALSE;
  ELSE
    stringCopy(rowForm, s, 0, commaPos-1);
    stringCopy(colForm, s, commaPos+1, rightPos-1);
    evaluateString(rowForm, curRow, curCol, v, st);
    IF st <> OK THEN
      cantEvalMsg(rowForm);
      RETURN FALSE;
    ELSE
      c.row := TRUNC(v);
      evaluateString(colForm, curRow, curCol, v, st);
      IF st <> OK THEN
        cantEvalMsg(colForm);
        RETURN FALSE;
      ELSE
        c.col := TRUNC(v);
        RETURN TRUE;
      END;
    END;
  END;
END cellRef1;

PROCEDURE cardFromString(s:ARRAY OF CHAR; VAR c:CARDINAL):BOOLEAN;
BEGIN
  StrToCard(s, c);
  IF NumConversionRes <> noError THEN
    message("Could not convert number");
  END;
  RETURN NumConversionRes = noError;
END cardFromString;

PROCEDURE cantEvalMsg(formString:ARRAY OF CHAR);
VAR msg:string160;
BEGIN
  Concat(msg, "Cannot evaluate ", formString);
  message(msg);
END cantEvalMsg;

BEGIN
END CommandProc.
ELSE
  operationStatus := RangeError;
END;
END clear;

```



```

PROCEDURE status(row, col: CARDINAL): Status;
BEGIN
  IF InRange(row, col) THEN
    operationStatus := OK;
    RETURN sheet[row, col].status;
  ELSE
    operationStatus := RangeError;
  END;
END status;

```

```

PROCEDURE maxRow(): CARDINAL;
BEGIN
  RETURN maxR;
END maxRow;

```

```

PROCEDURE maxCol(): CARDINAL;
BEGIN
  RETURN maxC;
END maxCol;

```

```

PROCEDURE recalculate;
VAR row, col: CARDINAL;
    val: REAL;
    stat: Status;
BEGIN
  FOR row := 1 TO maxR DO
    FOR col := 1 TO maxC DO
      WITH sheet[row, col] DO
        IF (status = OK) AND (NOT Formula.empty(form)) THEN
          evaluateFormula(form, row, col, val, stat);
          IF (stat <> status) OR (val <> value) THEN
            status := stat;
            value := val;
            displayCell(row, col);
          END;
        END;
      END;
    END;
  END;
  operationStatus := OK;
END recalculate;

```

```

BEGIN
END Spreadsheet.

```

cell2.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Cell;

(* This module implements the virtual spreadsheet. It uses the module
DiskIO for low-level I/O. Despite what it says in the article, I
do not check touchCount for overflow. *)

```

FROM CellList IMPORT cellList, nullCellList, freeCellList, nextCell,
    cellRow, cellCol, addToCellList;
IMPORT CellList;
FROM MyStorage2 IMPORT ALLOCATE, DEALLOCATE;
FROM Formula IMPORT formula, emptyFormula;
IMPORT Formula;
FROM Misc IMPORT fatal, assert;
FROM Spreadsheet IMPORT Status;
FROM DiskIO IMPORT DiskAddress, init, WriteReal, ReadReal, WriteString, clear,
    ReadString, WriteCard, ReadCard, freeDiskStorage, nullDiskAddress, empty,
    startWrite, endWrite, startRewrite, endRewrite, startRead, endRead,
    freeDiskAddress;
FROM StringStuff IMPORT string255;

```

(continued)

```

CONST nRows = 10;
      nCols = 10;

TYPE
  cellArray = ARRAY[0..nRows-1],[0..nCols-1] OF cell;
  cellArrayPtr = POINTER TO cellArray;
  cellChunkPtr = POINTER TO cellChunk;
  cellChunk = RECORD
    startRow, startCol: CARDINAL;
    down, right: cellChunkPtr;
    InMemory, dirty: BOOLEAN;
    touchedLast: CARDINAL;
    diskAddr: DiskAddress;
    cells: cellArrayPtr;
  END;

VAR sheet: cellChunkPtr;
    touchCount: CARDINAL;

PROCEDURE InitSheet;
BEGIN
  sheet := newCellChunk(1, 1, NIL, NIL);
  init; (* DiskIO *)
  touchCount := 0;
END InitSheet;

PROCEDURE clearSheet;
VAR rowCcp, colCcp, temp: cellChunkPtr;
BEGIN
  clear; (* DiskIO *)
  rowCcp := sheet;
  WHILE rowCcp <> NIL DO
    colCcp := rowCcp^.right;
    WHILE colCcp <> NIL DO
      temp := colCcp;
      colCcp := colCcp^.right;
      freeCellChunk(temp);
    END;
    temp := rowCcp;
    rowCcp := rowCcp^.down;
    freeCellChunk(temp);
  END;
  initSheet;
END clearSheet;

PROCEDURE getCell(row, col: CARDINAL; VAR c: cell);
VAR ccp: cellChunkPtr;
BEGIN
  INC(touchCount);
  IF findCellChunk(row, col, ccp) THEN
    IF NOT ccp^.InMemory THEN
      getCellsFromDisk(ccp);
    END;
    WITH ccp^ DO
      c := cells^[row-startRow, col-startCol];
      touchedLast := touchCount;
    END;
  ELSE
    clearCell(c);
  END;
END getCell;

PROCEDURE setCell(row, col: CARDINAL; c: cell);
VAR ccp: cellChunkPtr;
BEGIN
  INC(touchCount);
  IF NOT findCellChunk(row, col, ccp) THEN
    ccp := addCellChunk(row, col, ccp);
  END;
  IF NOT ccp^.InMemory THEN
    getCellsFromDisk(ccp);
  END;
  WITH ccp^ DO
    cells^[row-startRow, col-startCol] := c;
    touchedLast := touchCount;
    dirty := TRUE;
  END;
END setCell;

```



```
END;
END setCell;
```

```
PROCEDURE addCellChunk(row, col: CARDINAL; ccp: cellChunkPtr): cellChunkPtr;
VAR newCcp: cellChunkPtr;
BEGIN
  newCcp := newCellChunk(start(row, nRows), start(col, nCols), NIL, NIL);
  WITH ccp^ DO
    IF (row >= startRow) AND (row < startRow+nRows) THEN
      (* the new cell chunk belongs to the right of this one *)
      newCcp^.right := right;
      right := newCcp;
    ELSIF row >= startRow+nRows THEN
      (* the new cell chunk belongs below this one *)
      IF (down = NIL) OR (row < down^.startRow) THEN
        (* the new cell chunk belongs directly below *)
        newCcp^.down := down;
        down := newCcp;
      ELSE (* it belongs to the left of the one below *)
        assert((row >= down^.startRow) AND (row < down^.startRow+nRows),
          "addCellChunk: wrong chunk returned");
        newCcp^.right := down;
        newCcp^.down := down^.down;
        down := newCcp;
      END;
    ELSE
      fatal('addCellChunk: wrong cell chunk returned');
    END;
  END;
  RETURN newCcp;
END addCellChunk;
```

```
PROCEDURE start(rowOrCol, n: CARDINAL): CARDINAL;
(* computes start row or start col. *)
BEGIN
  RETURN ((rowOrCol-1) DIV n)*n + 1;
END start;
```

```
PROCEDURE findCellChunk(row, col: CARDINAL; VAR ccp: cellChunkPtr): BOOLEAN;
(* Returns TRUE if it finds the cell chunk containing the cell [row,col].
  If it returns FALSE, then ccp will point to the cell chunk just "before"
  where the right cell chunk should be. Before can mean just above,
  or just to the left, depending upon where the new cell chunk should be
  put. *)
BEGIN
  ccp := sheet;
  WHILE row >= ccp^.startRow+nRows DO
    WITH ccp^ DO
      IF (down = NIL) (* need to add a new chunk below this one *)
        OR (row < down^.startRow) (* need new chunk between this and next *)
        OR ((row < down^.startRow+nRows) AND
          (col < down^.startCol)) THEN (* need new chunk below & left *)
        RETURN FALSE;
      ELSE
        ccp := down;
      END;
    END;
  END;
  (* We are now on the correct row *)
  WHILE col >= ccp^.startCol+nCols DO
    IF (ccp^.right = NIL) OR (col < ccp^.right^.startCol) THEN
      RETURN FALSE;
    ELSE
      ccp := ccp^.right;
    END;
  END;
  RETURN TRUE;
END findCellChunk;
```

```
PROCEDURE doForAllCells(cp: cellProc);
VAR rowCcp, colCcp: cellChunkPtr;
BEGIN
  rowCcp := sheet;
  REPEAT
```

(continued)

```

colCcp:= rowCcp;
REPEAT
  doForCellChunk(colCcp, cp);
  colCcp := colCcp^.right;
UNTIL colCcp = NIL;
rowCcp := rowCcp^.down;
UNTIL rowCcp = NIL;
END doForAllCells;

```

```

PROCEDURE doForCellChunk(ccp:cellChunkPtr; cp:cellProc);
VAR row, col:CARDINAL;
BEGIN
  WITH ccp^ DO
    IF NOT inMemory THEN
      getCellsFromDisk(ccp);
    END;
    FOR row := startRow TO startRow+nRows-1 DO
      FOR col := startCol TO startCol+nCols-1 DO
        cp(row, col, cells^[row-startRow, col-startCol]);
      END;
    END;
  END;
END doForCellChunk;

```

```

PROCEDURE newCellChunk(startR, startC:CARDINAL; d,r:cellChunkPtr):cellChunkPtr;
VAR ccp:cellChunkPtr;
BEGIN

```

```

  NEW(ccp);
  WHILE ccp = NIL DO (* out of memory; throw out a chunk to disk *)
    putCellsOnDisk;
    NEW(ccp);
  END;
  WITH ccp^ DO
    cells := newCellArray();
    startRow := startR;
    startCol := startC;
    down := d;
    right := r;
    inMemory := TRUE;
    dirty := TRUE;
    touchedLast := 0;
    diskAddr := nullDiskAddress;
  END;
  RETURN ccp;
END newCellChunk;

```

```

PROCEDURE freeCellChunk(VAR ccp:cellChunkPtr);
BEGIN
  IF ccp^.inMemory THEN
    freeCellArray(ccp^.cells);
  END;
  freeDiskAddress(ccp^.diskAddr);
  DISPOSE(ccp);
  ccp := NIL;
END freeCellChunk;

```

```

PROCEDURE newCellArray():cellArrayPtr;
VAR cap:cellArrayPtr;
BEGIN
  NEW(cap);
  WHILE cap = NIL DO
    putCellsOnDisk;
    NEW(cap);
  END;
  clearCellArray(cap);
  RETURN cap;
END newCellArray;

```

```

PROCEDURE freeCellArray(VAR cap:cellArrayPtr);
VAR row, col:CARDINAL;
BEGIN
  FOR row := 0 TO nRows-1 DO
    FOR col := 0 TO nCols-1 DO
      WITH cap^[row, col] DO

```



```

        freeCellList(dependentCells);
        Formula.free(form);
    END;
END;
DISPOSE(cap);
cap := NIL;
END freeCellArray;

PROCEDURE clearCellArray(VAR cap:cellArrayPtr);
VAR row, col:CARDINAL;
BEGIN
    FOR row := 0 TO nRows-1 DO
        FOR col := 0 TO nCols-1 DO
            clearCell(cap^[row, col]);
        END;
    END;
END clearCellArray;

PROCEDURE clearCell(VAR c:cell);
BEGIN
    WITH c DO
        status := Empty;
        form := emptyFormula;
        dependentCells := nullCellList;
    END;
END clearCell;

(* Virtual memory stuff *)

PROCEDURE getCellsFromDisk(ccp:cellChunkPtr);
BEGIN
    WITH ccp^ DO
        cells := newCellArray(); (* This may result in a chunk being swapped out *)
        readCells(cells, diskAddr);
        inMemory := TRUE;
        dirty := FALSE;
    END;
END getCellsFromDisk;

PROCEDURE putCellsOnDisk;
VAR ccp:cellChunkPtr;
BEGIN
    ccp := findChunkToThrowOut();
    WITH ccp^ DO
        IF dirty THEN
            writeCells(ccp^.cells, ccp^.diskAddr);
        END;
        freeCellArray(cells);
        inMemory := FALSE;
    END;
END putCellsOnDisk;

PROCEDURE findChunkToThrowOut():cellChunkPtr;
(* Uses LRU to find chunk to discard. Looks through every cell chunk in
memory, returning the one accessed least recently.
Resets touchCount. *)
VAR rowCcp, ccp, lruCcp:cellChunkPtr;
BEGIN
    lruCcp := NIL; (* the least recently used chunk found *)
    rowCcp := sheet;
    WHILE rowCcp <> NIL DO
        ccp := rowCcp;
        WHILE ccp <> NIL DO
            WITH ccp^ DO
                IF inMemory AND
                    ((lruCcp = NIL) OR (touchedLast < lruCcp^.touchedLast)) THEN
                    lruCcp := ccp;
                END;
                ccp := right;
            END;
        END;
        rowCcp := rowCcp^.down;
    END;
END;

```

(continued)

```

assert(lruCcp <> NIL, "findChunkToThrowOut: no chunks in memory!");
(* This is possible if, say, there are so many cell chunks allocated
   that the storage for them, even when the cells parts are on disk,
   exceeds memory capacity. *)
RETURN lruCcp;
END findChunkToThrowOut;

PROCEDURE writeCells(cap:cellArrayPtr; VAR da:DiskAddress);
VAR s:string255;
    row, col:CARDINAL;
BEGIN
    IF empty(da) THEN
        startWrite;
    ELSE
        startRewrite(da);
    END;
    FOR row := 0 TO nRows-1 DO
        FOR col := 0 TO nCols-1 DO
            WITH cap^[row,col] DO
                WriteReal(value);
                WriteCard(CARDINAL(status));
                Formula.toString(form, s);
                WriteString(s);
                writeCellList(dependentCells);
            END;
        END;
    END;
    IF empty(da) THEN
        da := endWrite();
    ELSE
        da := endRewrite();
    END;
END writeCells;

PROCEDURE readCells(cap:cellArrayPtr; da:DiskAddress);
VAR s:string255;
    row, col, temp:CARDINAL;
BEGIN
    startRead(da);
    FOR row := 0 TO nRows-1 DO
        FOR col := 0 TO nCols-1 DO
            WITH cap^[row,col] DO
                ReadReal(value);
                ReadCard(temp);
                status := Status(temp);
                ReadString(s);
                form := Formula.toFormula(s);
                readCellList(dependentCells);
            END;
        END;
    END;
    endRead;
END readCells;

PROCEDURE writeCellList(cl:cellList);
(* Writes a cell list by first writing its length, then the pairs of row,col *)
VAR clist:cellList;
    len:CARDINAL;
BEGIN
    clist := cl;
    len := 0;
    WHILE NOT CellList.empty(clist) DO
        INC(len);
        clist := nextCell(clist);
    END;
    WriteCard(len);
    clist := cl;
    WHILE NOT CellList.empty(clist) DO
        WriteCard(cellRow(clist));
        WriteCard(cellCol(clist));
        clist := nextCell(clist);
    END;
END writeCellList;

PROCEDURE readCellList(VAR cl:cellList);
VAR len, i, row, col:CARDINAL;
BEGIN

```



```

ReadCard(len);
cl := nullCellList;
IF len <> 0 THEN
  FOR i := 1 TO len DO
    ReadCard(row);
    ReadCard(col);
    addToCellList(cl, row, col);
  END;
END;
END readCellList;

(* For debugging only.
PROCEDURE writeChunk(ccp:cellChunkPtr);
BEGIN
  Write(['');
  MyTerminal.WriteCard(ccp^.startRow, 0);
  Write(',');
  MyTerminal.WriteCard(ccp^.startCol, 0);
  Write(']');
END writeChunk;
*)

BEGIN
END Cell.

```

commandp.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE CommandProc;

FROM Formula IMPORT formula;

EXPORT QUALIFIED command, CommandType, readCommand;

TYPE CommandType = (CellRef, SetValue, SetFormula, Left, Right, Up, Down,
  NewCorner, Precision, ColWidth, Automatic, Manual,
  Recalc, Copy, Clear, Quit);

command =
  RECORD
    CASE type:CommandType OF
      CellRef, NewCorner, Copy: row, col:CARDINAL;
      | SetValue: value:REAL;
      | SetFormula: form: formula;
      | Precision: precision:CARDINAL;
      | ColWidth: colWidth:CARDINAL;
    END;
  END;

PROCEDURE readCommand(VAR c:command; crRow, crCol:CARDINAL);

END CommandProc.

```

cell1st.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

IMPLEMENTATION MODULE CellList;

(* Implements cell lists, and finding dependencies in formulas.
*)

FROM StringStuff IMPORT string160, findChar;
FROM Formula IMPORT formula;
IMPORT Formula;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM MyTerminal IMPORT WriteString, WriteCard, WriteLnString;
FROM Evaluator IMPORT refexpr;

```

(continued)

```
FROM Spreadsheet IMPORT Status;
FROM StringOps IMPORT Delete;
```

```
TYPE
```

```
  cellList = POINTER TO cellNode;
  cellNode = RECORD
    row, col: CARDINAL;
    next: cellList;
  END;
```

```
VAR string: string160;
    cRow, cCol: CARDINAL;
```

```
PROCEDURE empty(cl: cellList): BOOLEAN;
BEGIN
  RETURN cl = NIL;
END empty;
```

```
PROCEDURE nextCell(cl: cellList): cellList;
BEGIN
  RETURN cl^.next;
END nextCell;
```

```
PROCEDURE cellRow(cl: cellList): CARDINAL;
BEGIN
  RETURN cl^.row;
END cellRow;
```

```
PROCEDURE cellCol(cl: cellList): CARDINAL;
BEGIN
  RETURN cl^.col;
END cellCol;
```

```
PROCEDURE initFindDep(curRow, curCol: CARDINAL; f: formula);
BEGIN
  Formula.toString(f, string);
  cRow := curRow;
  cCol := curCol;
END initFindDep;
```

```
PROCEDURE nextDep(VAR row, col: CARDINAL): BOOLEAN;
(* assumes syntactically correct formula *)
VAR lbPos, commaPos, rbPos: CARDINAL;
    vCol, vRow: REAL;
    s: Status;
```

```
BEGIN
  IF (NOT findChar(string, '[', lbPos)) OR
     (NOT findChar(string, ',', commaPos)) OR
     (NOT findChar(string, ']', rbPos)) THEN
    RETURN FALSE;
  ELSE
    INC(lbPos);
    refexpr(string, lbPos, vRow, s, cRow);
    IF s <> OK THEN
      WriteLnString("nextDep: s <> OK");
    END;
    INC(commaPos);
    refexpr(string, commaPos, vCol, s, cCol);
    IF s <> OK THEN
      WriteLnString("nextDep: s <> OK");
    END;
    row := TRUNC(vRow);
    col := TRUNC(vCol);
    Delete(string, 0, rbPos+1);
    RETURN TRUE;
  END;
END nextDep;
```

```
PROCEDURE addToCellList(VAR cl: cellList; row, col: CARDINAL);
VAR newcl: cellList;
BEGIN
  IF NOT member(row, col, cl) THEN
    NEW(newcl);
    newcl^.row := row;
    newcl^.col := col;
    newcl^.next := cl;
```



```

    cl := newcl;
END;
END addToCellList;

PROCEDURE member(row, col: CARDINAL; cl: cellList): BOOLEAN;
BEGIN
    WHILE cl <> NIL DO
        IF (row = cl^.row) AND (col = cl^.col) THEN
            RETURN TRUE;
        ELSE
            cl := cl^.next;
        END;
    END;
    RETURN FALSE;
END member;

PROCEDURE removeFromCellList(VAR cl: cellList; row, col: CARDINAL);
(* Writes error if not found. *)
VAR temp, prev: cellList;
BEGIN
    temp := cl;
    prev := NIL;
    WHILE temp <> NIL DO
        IF (temp^.row = row) AND (temp^.col = col) THEN
            IF temp = cl THEN
                cl := cl^.next;
            ELSE
                prev^.next := temp^.next;
            END;
            DISPOSE(temp);
            RETURN;
        ELSE
            prev := temp;
            temp := temp^.next;
        END;
    END;
    END;
    WriteString("removeFromCellList: ");
    WriteCard(row, 3);
    WriteCard(col, 3);
    WriteLnString(" not found");
END removeFromCellList;

PROCEDURE freeCellList(VAR cl: cellList);
VAR temp: cellList;
BEGIN
    WHILE cl <> NIL DO
        temp := cl;
        cl := cl^.next;
        DISPOSE(temp);
    END;
END freeCellList;

BEGIN
    nullCellList := NIL;
END CellList.

```

cell.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE Cell;

FROM CellList IMPORT cellList;
FROM Formula IMPORT formula;
FROM Spreadsheet IMPORT Status;

EXPORT QUALIFIED cell, getCell, setCell, doForAllCells,
    initSheet, clearSheet, cellProc;

TYPE
    cell = RECORD

```

(continued)

```

        value: REAL;
        form: formula;
        status: Status;
        dependentCells: cellList;
    END;
    cellProc = PROCEDURE(CARDINAL, CARDINAL, VAR cell);

```

```

PROCEDURE initSheet;

PROCEDURE clearSheet;

PROCEDURE getCell(row, col: CARDINAL; VAR c: cell);

PROCEDURE setCell(row, col: CARDINAL; c: cell);

PROCEDURE doForAllCells(cp: cellProc);

END Cell.

```

cell1.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE Cell;

```

FROM CellList IMPORT cellList, nullCellList, freeCellList;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM Formula IMPORT formula, emptyFormula;
IMPORT Formula;
FROM Misc IMPORT fatal, assert;
FROM Spreadsheet IMPORT Status;

```

```

CONST nRows = 10;
      nCols = 10;

```

TYPE

```

    cellArray = ARRAY[0..nRows-1],[0..nCols-1] OF cell;
    cellChunkPtr = POINTER TO cellChunk;
    cellChunk = RECORD
        startRow, startCol: CARDINAL;
        cells: cellArray;
        down, right: cellChunkPtr;
    END;

```

```

VAR sheet: cellChunkPtr;

```

```

PROCEDURE initSheet;
BEGIN
    sheet := newCellChunk(1, 1, NIL, NIL);
END initSheet;

```

```

PROCEDURE clearSheet;
VAR rowCcp, colCcp, temp: cellChunkPtr;
BEGIN
    rowCcp := sheet;
    WHILE rowCcp <> NIL DO
        colCcp := rowCcp^.right;
        WHILE colCcp <> NIL DO
            temp := colCcp;
            colCcp := colCcp^.right;
            freeCellChunk(temp);
        END;
        temp := rowCcp;
        rowCcp := rowCcp^.down;
        freeCellChunk(temp);
    END;
    initSheet;
END clearSheet;

```

```

PROCEDURE getCell(row, col: CARDINAL; VAR c: cell);
VAR ccp: cellChunkPtr;

```



```

BEGIN
  IF findCellChunk(row, col, ccp) THEN
    WITH ccp^ DO
      c := cells[row-startRow, col-startCol];
    END;
  ELSE
    clearCell(c);
  END;
END getCell;

PROCEDURE setCell(row, col: CARDINAL; c: cell);
VAR ccp: cellChunkPtr;
BEGIN
  IF NOT findCellChunk(row, col, ccp) THEN
    ccp := addCellChunk(row, col, ccp);
  END;
  WITH ccp^ DO
    cells[row-startRow, col-startCol] := c;
  END;
END setCell;

PROCEDURE addCellChunk(row, col: CARDINAL; ccp: cellChunkPtr): cellChunkPtr;
VAR newCcp: cellChunkPtr;
BEGIN
  newCcp := newCellChunk(start(row, nRows), start(col, nCols), NIL, NIL);
  WITH ccp^ DO
    IF (row >= startRow) AND (row < startRow+nRows) THEN
      (* the new cell chunk belongs to the right of this one *)
      newCcp^.right := right;
      right := newCcp;
    ELSIF row >= startRow+nRows THEN
      (* the new cell chunk belongs below this one *)
      IF col < down^.startCol THEN
        (* the new cell chunk belongs to the left of the one below *)
        assert((row >= down^.startRow) AND (row < down^.startRow+nRows),
          "addCellChunk: wrong chunk returned");
        newCcp^.right := down;
        newCcp^.down := down^.down;
        down := newCcp;
      ELSE (* it just belongs directly below *)
        newCcp^.down := down;
        down := newCcp;
      END;
    ELSE
      fatal('addCellChunk: wrong cell chunk returned');
    END;
  END;
  RETURN newCcp;
END addCellChunk;

PROCEDURE start(rowOrCol, n: CARDINAL): CARDINAL;
(* computes start row or start col. *)
BEGIN
  RETURN ((rowOrCol-1) DIV n)*n + 1;
END start;

PROCEDURE findCellChunk(row, col: CARDINAL; VAR ccp: cellChunkPtr): BOOLEAN;
(* Returns TRUE if it finds the cell chunk containing the cell [row,col].
  If it returns FALSE, then ccp will point to the cell chunk just "before"
  where the right cell chunk should be. Before can mean just above,
  or just to the left, depending upon where the new cell chunk should be
  put. *)
BEGIN
  ccp := sheet;
  WHILE row >= ccp^.startRow+nRows DO
    WITH ccp^ DO
      IF (down = NIL) (* need to add a new chunk below this one *)
        OR (row < down^.startRow) (* need new chunk between this and next *)
        OR ((row < down^.startRow+nRows) AND
          (col < down^.startCol)) THEN (* need new chunk below & left *)
        RETURN FALSE;
      ELSE
        ccp := down;
      END;
    END;
  END;
END;

```

(continued)

```

END;
(* We are now on the right row *)
WHILE col >= ccp^.startCol+nCols DO
  IF (ccp^.right = NIL) OR (col < ccp^.right^.startCol) THEN
    RETURN FALSE;
  ELSE
    ccp := ccp^.right;
  END;
END;
RETURN TRUE;
END findCellChunk;

PROCEDURE doForAllCells(cp:cellProc);
VAR rowCcp, colCcp:cellChunkPtr;
BEGIN
  rowCcp := sheet;
  REPEAT
    colCcp := rowCcp;
    REPEAT
      doForCellChunk(colCcp, cp);
      colCcp := colCcp^.right;
    UNTIL colCcp = NIL;
    rowCcp := rowCcp^.down;
  UNTIL rowCcp = NIL;
END doForAllCells;

PROCEDURE doForCellChunk(ccp:cellChunkPtr; cp:cellProc);
VAR row, col:CARDINAL;
BEGIN
  WITH ccp^ DO
    FOR row := startRow TO startRow+nRows-1 DO
      FOR col := startCol TO startCol+nCols-1 DO
        cp(row, col, cells[row-startRow, col-startCol]);
      END;
    END;
  END;
END doForCellChunk;

PROCEDURE newCellChunk(startR, startC:CARDINAL; d,r:cellChunkPtr):cellChunkPtr;
VAR ccp:cellChunkPtr;
BEGIN
  NEW(ccp);
  WITH ccp^ DO
    startRow := startR;
    startCol := startC;
    down := d;
    right := r;
    clearCellChunk(ccp);
  END;
  RETURN ccp;
END newCellChunk;

PROCEDURE freeCellChunk(VAR ccp:cellChunkPtr);
VAR row, col:CARDINAL;
BEGIN
  FOR row := 0 TO nRows-1 DO
    FOR col := 0 TO nCols-1 DO
      WITH ccp^.cells[row, col] DO
        freeCellList(dependentCells);
        Formula.free(form);
      END;
    END;
  END;
  DISPOSE(ccp);
  ccp := NIL;
END freeCellChunk;

PROCEDURE clearCellChunk(VAR ccp:cellChunkPtr);
VAR row, col:CARDINAL;
BEGIN
  FOR row := 0 TO nRows-1 DO
    FOR col := 0 TO nCols-1 DO
      clearCell(ccp^.cells[row, col]);
    END;
  END;

```



```

END;
END clearCellChunk;

PROCEDURE clearCell(VAR c:cell);
BEGIN
  WITH c DO
    status := Empty;
    form := emptyFormula;
    dependentCells := nullCellList;
  END;
END clearCell;

BEGIN
END Cell.

```

misc.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

IMPLEMENTATION MODULE Misc;

FROM MyTerminal IMPORT pause, WriteLnString;

PROCEDURE fatal(msg:ARRAY OF CHAR);
BEGIN
  WriteLnString(msg);
  pause('Hit any key to die--');
  HALT;
END fatal;

PROCEDURE assert(test:BOOLEAN; msg:ARRAY OF CHAR);
BEGIN
  IF NOT test THEN
    fatal(msg);
  END;
END assert;

PROCEDURE max(a, b:CARDINAL):CARDINAL;
BEGIN
  IF a > b THEN
    RETURN a;
  ELSE
    RETURN b;
  END;
END max;

PROCEDURE min(a, b:CARDINAL):CARDINAL;
BEGIN
  IF a < b THEN
    RETURN a;
  ELSE
    RETURN b;
  END;
END min;

BEGIN
END Misc.

```

misc.def

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

```

DEFINITION MODULE Misc;

EXPORT QUALIFIED fatal, assert, max, min;

```

(continued)

```

PROCEDURE fatal(msg:ARRAY OF CHAR);
(* Prints the message, does a pause, and HALTs. *)

PROCEDURE assert(test:BOOLEAN; msg:ARRAY OF CHAR);
(* If test is FALSE, fatal is called with msg. Else, nothing. *)

PROCEDURE max(a, b:CARDINAL):CARDINAL;

PROCEDURE min(a, b:CARDINAL):CARDINAL;

END Misc.

```

stringst.mod

Programming Project: "Build a Spreadsheet Program," by
Jonathan Amsterdam. July, page 96.

IMPLEMENTATION MODULE StringStuff;

```

PROCEDURE charCap(ch:CHAR):CHAR;
BEGIN
  IF (ch >= 'a') AND (ch <= 'z') THEN
    RETURN CAP(ch);
  ELSE
    RETURN ch;
  END;
END charCap;

PROCEDURE stringCap(VAR s:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  FOR i := 0 TO stringLen(s) DO
    s[i] := charCap(s[i]);
  END;
END stringCap;

PROCEDURE stringLen(VAR s:ARRAY OF CHAR):CARDINAL;
VAR i:CARDINAL;
BEGIN
  FOR i := 0 TO HIGH(s) DO
    IF s[i] = 0C THEN
      RETURN i;
    END;
  END;
  RETURN HIGH(s)+1;
END stringLen;

PROCEDURE stringAssign(VAR dest:ARRAY OF CHAR; source:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  i := 0;
  LOOP
    IF i > HIGH(dest) THEN
      EXIT;
    ELSIF i > HIGH(source) THEN
      dest[i] := 0C;
      EXIT;
    ELSE
      dest[i] := source[i];
    END;
    INC(i);
  END;
END stringAssign;

PROCEDURE stringCopy(VAR dest, source:ARRAY OF CHAR; from, to:CARDINAL);
VAR i, num:CARDINAL;
BEGIN
  IF from <= to THEN
    num := to-from+1;
    FOR i := 0 TO num-1 DO
      IF i > HIGH(dest) THEN

```



```

        RETURN;
    ELSIF i+from > HIGH(source) THEN
        dest[i] := 0C;
        RETURN;
    ELSE
        dest[i] := source[i+from];
    END;
END;
IF num <= HIGH(dest) THEN
    dest[num] := 0C;
END;
ELSE
    dest[0] := 0C;
END;
END stringCopy;

PROCEDURE stringEqual(s1, s2:ARRAY OF CHAR):BOOLEAN;
VAR i:CARDINAL;
BEGIN
    FOR i := 0 TO HIGH(s1) DO
        IF i > HIGH(s2) THEN
            RETURN s1[i] = 0C;
        ELSIF s1[i] <> s2[i] THEN
            RETURN FALSE;
        ELSIF s1[i] = 0C THEN
            RETURN TRUE;
        END;
    END;
    RETURN TRUE;
END stringEqual;

PROCEDURE deleteChar(VAR s:ARRAY OF CHAR; pos:CARDINAL);
VAR i:CARDINAL;
BEGIN
    FOR i := pos TO HIGH(s)-1 DO
        s[i] := s[i+1];
    END;
END deleteChar;

PROCEDURE insertChar(ch:CHAR; VAR s:ARRAY OF CHAR; pos:CARDINAL);
VAR i:CARDINAL;
BEGIN
    FOR i := stringLen(s)-1 TO pos BY -1 DO
        s[i+1] := s[i];
    END;
    s[pos] := ch;
END insertChar;

PROCEDURE findChar(s:ARRAY OF CHAR; ch:CHAR; VAR pos:CARDINAL):BOOLEAN;
VAR i, len:CARDINAL;
BEGIN
    len := stringLen(s);
    IF len = 0 THEN
        RETURN FALSE;
    ELSE
        FOR i := 0 TO len-1 DO
            IF s[i] = ch THEN
                pos := i;
                RETURN TRUE;
            END;
        END;
        RETURN FALSE;
    END;
END findChar;

BEGIN
END StringStuff.

```

visc.bas

"Small-Scale Engineering Applications," by J. Neil Stone. July, page 253.

```

10 REM ESTIMATE VISCOSITY - May 85 - VISC/BAS
20 REM Final, Sept 2nd, 1985
30 REM Edited for BYTE Dec 85
40 GOTO 160
50 DATA"***** VISCOSITIES *****"
60 DATA"* OF LIQUIDS AND MIXTURES  *"
70 DATA" by"
80 DATA" J.Neil Stone"
90 DATA" Ledge Engineering Inc"
100 DATA" 179 Lansdowne Avenue"
110 DATA" Kingsville, Ontario, N9Y 3J2"
120 DATA" "
130 DATA" **** May 1985 ****"
140 CLS:FOR J=1 TO 9:READ Y:PRINT@ FN LC(NR/2-5+J,NS/2-17),Y:NEXT
150 RETURN
160 CLEAR 1500
170 DEFINTJ-L,N:DEFSTRX-Z
180 DIM
Y,J,K,NS,KP,Y%,LC,KQ,AT,KC,N,UT,A,J9,ET,TR,TL,J8,J$,KS,RD,NL,NC,F2,NR,V,NM
'simple vars
190 POKE 16409,1 'Set caps mode
200 NS=64:NR=16:LF=1 'screen parameters for TRS80
210 'NS=80:NR=24:LF=128 'screen parameters for 80x24 (IBM)
220 DEF FN LC(J,K)=(J-1)*NS+K-1
230 GOSUB 50 'display title
240 DIM A(2,92),AX(10),AY(10),YT(8),Y(8),AV(10),AM(10),YN(10),YM(3),MC(10)
'arrays
250 REM Set up message strings
260 Y(1)="Molecular weight":Y(2)="Critical temp, K":Y(3)="Critical press,
atm":Y(4)="Critical vol, cc/mol":Y(5)="Acentric factor":Y(8)="Melting point,
deg C":Y(6)="Density, g/cc (SG)":Y(7)="Temp. of density measure, deg C"
270 YM(1)="Przedziecki & Sridhar":YM(2)="Stiel & Thodos":YM(3)="AIChE Data
Prediction Manual"
280 Z="---":Z1=CHR$(91)+CHR$(10)+CHR$(13)+"Xx"+CHR$(9):Z3=" ":Z4="<--
":Z6="-.0123456789":Z2=Z6+Z1:Z5=Z6+"ED"+CHR$(13)+CHR$(8)
290 Z8="USE UP, DOWN, RT ARROWS TO SELECT ENTRY. <X> TO EXIT.":Z9="USE "+Z4+"
TO CORRECT. PRESS <ENTER> WHEN DONE":ZC=STRING$(17," ")
300 Y1="###.###":Y2="###":Y3="###.###":Y4="####.####":Y5="##.####^ ^ ^ ^"
310 REM Set up constants
320 C1=273.16:C2=2/7:C3=1E-10
330 ON ERROR GOTO 7190
340 GOTO 1000 'to main prog
390 REM All-purpose screen for 1,2 or 3 col numeric entry
400 REM Screen layout
410 CLS:PRINT TAB((NS-LEN(ZA))/2) ZA
420 PRINT STRING$(NS-1,"=")
430 FOR J=1 TO NT:J$=RIGHT$(STR$(J),1)
440 PRINT YT(J);
450 IF INSTR(ZD,J$) THEN PRINT:GOTO 490 ELSE IF INSTR(ZG,J$) THEN 480
460 IF J8>2 THEN PRINT TAB(NS-46);A(0,KS+J-1);
470 IF J8>1 THEN PRINT TAB(NS-30);A(1,KS+J-1);
480 PRINT TAB(NS-14);A(2,KS+J-1)
490 NEXT
500 REM Display data as SP to prevent overflows
510 PRINT@ FN LC(14,1),STRING$(NS-1,95)
520 PRINT@ FN LC(15,1),CHR$(30);ZB;
530 REM Begin display
540 KQ=NS-17:KP=KL+1:J9=6:GOTO 600
550 PRINT@ FN LC(KP+1,KQ+1),Z;
560 PRINT@ FN LC(16,1),CHR$(30);Z8;
570 Y$=INKEY$:IF Y$="" THEN 570
580 IF INSTR(Z2,Y$)=0 THEN 570
590 J9=INSTR(Z1,Y$)
600 IF J9=0 THEN 760 ELSE PRINT@ FN LC(KP+1,KQ+1),Z3;
610 ON J9 GOTO 640,620,620,630,630,670
620 J9=1:GOTO 650
630 RETURN
640 J9=-1
650 KP=KP+J9:IF KP>KM+1 THEN KP=KL+1 ELSE IF KP<KL+1 THEN KP=KM+1

```



```

660 GOTO 710
670 ON J8 GOTO 710,700
680 IF KQ=NS-49 OR KQ=NS-33 THEN KQ=KQ+16 ELSE KQ=NS-49
690 GOTO 710
700 IF KQ=NS-33 THEN KQ=NS-17 ELSE KQ=NS-33
710 ZF=STR$(KP-1):ZF=RIGHT$(ZF,LEN(ZF)-1)+", "
720 IF INSTR(ZD,ZF) THEN IF J9=6 THEN J9=1:GOTO 650 ELSE 650
730 IF INSTR(ZG,ZF) THEN KQ=NS-17
740 GOTO 550
750 REM Data entry section
760 PRINT@FN LC(16,1),CHR$(30);Z9;
770 KC=1:ZE=Y$
780 PRINT@ FN LC(KP+1,KQ+1),ZC::PRINT@ FN LC(KP+1,KQ+5),ZE;CHR$(95);
790 Y$=INKEY$:IF Y$="" THEN 790
800 Y%=INSTR(Z5,Y$):IF Y%=0 THEN 790
810 IF Y%>15 THEN ON Y%-15 GOTO 880,860
820 KC=KC+1
830 ZE=ZE+Y$
840 PRINT@ FN LC(KP+1,KQ+KC+4),Y$;CHR$(95);
850 GOTO 790
860 KC=KC-1:IF KC<=0 THEN KC=0:ZE="" ELSE ZE=LEFT$(ZE,KC)
870 GOTO 780
880 KE=(KQ-NS+49)/16
890 A(KE,KP+KS-2)=VAL(ZE)
900 PRINT@ FN LC(KP+1,KQ+1),ZC::PRINT@ FN LC(KP+1,KQ+4),CSNG(VAL(ZE));
910 ZE="":J9=1
920 IF KQ=NS-17 THEN KP=KP+1
930 IF KP>KM+1 THEN KP=KL+1
940 GOTO 670
990 REM Main program
1000 GOSUB 8000 'introductory text
1010 GOSUB 6620 'title etc
1020 GOSUB 6080 'main menu
1030 ON NM GOSUB 1250,1250,1500 'goto correct routine
1040 IF NC<>6 THEN 1060
1050 ON ERROR GOTO 0:END 'end of program
1060 FOR K=0 TO 2:FOR J=1 TO 8:A(K,J)=0:NEXTJ,K:LP=0
1070 IF NC=5 THEN JP=0:LT=0
1080 ON NC-3 GOTO 1020,1010
1240 REM Viscosity prediction by both methods
1250 IF NE THEN 1290 'go to input if returned by error
1260 GOSUB 8270 'intro text
1270 GOSUB 2000 'name of substance
1280 GOSUB 8550 'load data from file, if any
1290 GOSUB 2030 'enter phys props
1300 GOSUB 2150 'calc for substance
1310 GOSUB 6170 'calculation options
1320 IF TN=0 THEN GOSUB 2360:GOSUB 4500 ELSE GOSUB 2500
'calc for temp of interest
1330 GOSUB 6400 'what next
1340 ON NC GOTO 1360,1310,1370
1350 RETURN
1360 CLS:GOSUB 5000:GOTO 1330
1370 FOR K=0 TO 2:FOR J=1 TO 8:A(K,J)=0:NEXTJ,K:LP=0:GOTO 1270
1490 REM Mixture routine
1500 IF NE THEN 1540 'to entry if return on error
1510 GOSUB 8470 'intro blurb
1520 GOSUB 3000 'get names of materials
1530 IF NL=2 THEN GOSUB 3120 'range of calcs?
1540 GOSUB 3250 'data in
1550 GOSUB 3490 'estimate missing viscosities
1560 GOSUB 3750 'viscosity of mixture/print
1570 GOSUB 6510 'what next?
1580 ON NC GOTO 1610,1530,1650
1590 RETURN
1600 REM Hardcopy results
1610 IF AL THEN CLS:PRINT"Too late! Cannot print results for range now.":GOSUB
7070:GOTO 1570
1620 CLS:GOSUB 5410 'hardcopy results
1630 GOTO 1570
1640 REM Clean out data
1650 AL=0:FOR J=1 TO NL:FOR K=0 TO 2:A(K,J)=0:NEXT K,J:GOTO 1520
1990 REM Input material data
2000 CLS:KS=1
2010 INPUT"Name of Substance";YB

```

(continued)

July

```
2020 RETURN
2030 CLS:NE=0 'reset error flag
2040 KS=1:KL=1:KM=8:ZA="PROPERTIES OF "+YB
2050 ZB="":ZD="":ZG="":J8=1
2060 IF NM=2 THEN KM=7 'no f. pt needed for St & Th
2070 FOR J=1 TO KM:YT(J)=Y(J):NEXT 'instal titles
2080 NT=9-NM 'no of titles
2090 GOSUB 410 'to data input
2100 GOSUB 2600 'check input data valid
2110 IF NE THEN NE=0:GOTO 2090 ELSE GOSUB 8850
2120
MW=A(2,1):TC=A(2,2):PC=A(2,3):VC=A(2,4):W=A(2,5):TF=A(2,8)+C1:SG=A(2,6):TD=A(2,7)+C1 'get data in equation variables
2130 RETURN
2140 REM Calculate basic parameters
2150 CLS
2160 TR=TD/TC:FZ=.29056-.08775*W 'Yamada & Gunn, 1973
2170 GOSUB 2290
2180 VS=MW/SG/F2 'calc scaling factor
2190 IF NM=2 THEN GOTO 2320 'Stiel and Thodos
2200 REM Pr and Sr method
2210 TR=TF/TC
2220 GOSUB 2290
2230 VM=F2*VS
2240 VZ=.0085*TC*W-2.02+VM/((.342*TR+.894))
2250 F1=4.27+.032*MW-.077*PC+.014*TF-3.82*TR
2260 B=.33*VC/F1-1.12
2270 RETURN
2280 REM Calc f2(t)
2290 F2=FZ^((1-TR)^C2) 'Yamada & Gunn 1973
2300 RETURN
2310 REM Stiel and Thodos parameter
2320 ES=TC^(1/6)
2330 ES=ES/SQR(MW)/(PC^(2/3))
2340 RETURN
2350 REM Calc for specific temp
2360 TR=(TL+C1)/TC
2370 GOSUB 2290:V=F2*VS
2380 IF NM=2 THEN GOTO 2420 'to Stiel and Thodos
2390 ET=B*(V/VZ-1):ET=1/ET
2400 GOTO 2480
2410 REM Stiel and Thodos calc
2420 IF TR<1.5 THEN UB=34E-05*(TR^.94) ELSE UB=17.78E-05*((4.58*TR-1.67)^(5/8)) 'calc low press visc
2430 RD=VC/V
2440 UT=(((.0093324*RD-.040758)*RD+.058533)*RD+.023364)*RD+.1023
2450 UT=UT*UT*UT*UT
2460 UT=UT-1E-04+UB
2470 ET=UT/ES
2480 RETURN
2490 REM Calc for range of temps
2500 CLS
2510 FOR TL=TS TO TE STEP TN
2520 GOSUB 2360
2530 GOSUB 5170
2540 GOSUB 4650
2550 NEXT
2560 TL=TL-TN 'restore last temp calc'd
2570 IF N THEN GOSUB 7070
2580 RETURN
2590 REM Check validity of input data
2600 FOR J=KS TO KS+5
2610 IF A(2,J)>0 THEN 2630
2620 CLS:PRINT Y(J-KS+1);" cannot be zero or negative.":NE=1
2630 NEXT
2640 IF NE THEN PRINT:PRINT"Program will return to data entry. Please correct input.":GOSUB 7070
2650 RETURN
2990 REM Initial input for mixtures
3000 CLS:JP=0
3010 NL=0 'initialise - NL no of comps
3020 NL=NL+1:YN(NL)="" :MC(NL)=0:PRINT "Name of component no: ";NL " - just
<ENTER> if no more components.":INPUT YN(NL)
3030 IF YN(NL)<>" " THEN IF NL<10 GOTO 3020
3040 NL=NL-1:IF NL<2 THEN PRINT "Must have at least two components.":GOTO 3020
3050 CLS
```



```

3060 PRINT "Are your compositions in wt% or mol% - type <W> or <M>"
3070 YR="WwMm":GOSUB 7040
3080 MX=(KA>2) 'mx true if mol%
3090 IF MX THEN YU="mol%" ELSE YU="wt%"
3100 RETURN
3110 REM Calculate for range
3120 CLS:AL=0:MP=0:PRINT "Do you want to automatically calculate viscosities
for a range of compositions Y/N?"
3130 GOSUB 7030
3140 IF KA>2 THEN 3230
3150 PRINT "Minimum value of ";YU" for ";YN(1):INPUT AL
3160 IF (AL<0) OR (AL>100) THEN PRINT YU;"must be in range 0-100":GOTO 3150
3170 PRINT "Maximum value of ";YU" for ";YN(1):INPUT AH
3180 IF (AH<0) OR (AH>100) OR (AH<AL) THEN PRINT YU;"must be in range 0-100
and max must be bigger than min.":GOTO 3170
3190 PRINT "Increments of ";YU" for ";YN(1):INPUT AD
3200 PRINT:PRINT "Do you want hardcopy of the results as they are done, Y/N?"
3210 GOSUB 7030
3220 MP=(KA<3) 'true if hardcopy needed
3230 RETURN
3240 REM Initial data entry for mixtures
3250 FOR J=1 TO NL:IF LEN(YN(J))>NS-51 THEN YT(J+1)=LEFT$(YN(J),NS-51) ELSE
YT(J+1)=YN(J)
3260 NEXT
3270 YT(1)=STRING$(NS-46," ")+"Visc - cp"+STRING$(7," ")+"Mol.Wt"+STRING$(10,"
")+YU
3280 ZA="MIXTURE VISCOSITY INPUT DATA"
3290 ZB="LEAVE ANY UNKNOWN DATA AS 0 - MW NOT NEEDED IF USING MOL%"
3300 ZD="1,";ZG="";J8=3:NT=NL+1:KL=2:KM=NL+1:KS=0
3310 IF AL THEN A(2,1)=AL:A(2,2)=100-AL
3320 GOSUB 410 'screen routine
3330 GOSUB 3390 'check data
3340 IF NE THEN NE=0:GOTO 3320 'return if data check not OK
3350 FOR J=1 TO NL:AV(J)=A(0,J):AM(J)=A(1,J):AX(J)=A(2,J):NEXT 'put data into
working vars; av=visc, am=mw, ax=frn
3360 TL=-1000 'flag to show temp not set
3370 RETURN
3380 REM Check screen input for mixtures
3390 CLS:NE=0:A=0
3400 FOR J=1 TO NL
3410 IF NOT MX THEN IF A(1,J)<=0 THEN PRINT"Molecular wt for ";YN(J);"
required":NE=1
3420 IF NL<>2 OR AL=0 THEN IF A(2,J)<0 OR A(2,J)>100 THEN PRINT YU;" of
";YN(J);" out of range":NE=1
3430 A=A+A(2,J)
3440 NEXT
3450 IF A>100 OR A<100 THEN PRINT"Components do not add to 100 ";YU:NE=1
3460 IF NE THEN GOSUB 7070
3470 RETURN
3480 REM Estimate viscosity of missing items
3490 CLS:NK=0 'nk is temperature flag
3500 FOR LM=1 TO NL 'lm is main counter
3510 IF AV(LM) THEN 3720 'skip if visc given
3520 MC(LM)=1:PRINT "Estimating viscosity of ";YN(LM)
3530 PRINT:PRINT "Is ";YN(LM):PRINT "a non-associating compound, Y/N?";
3540 GOSUB 7030
3550 IF KA<3 THEN NM=1 ELSE NM=2
3560 YB=YN(LM):KS=LM*8+4:A(2,KS)=AM(LM):GOSUB 8550 'check for data on disk
3570 CLS:NE=0 'reset error flag
3580 KL=1:KM=8:ZA="PROPERTIES OF "+YB
3590 ZB="";ZD="";ZG="";J8=1
3600 IF NM=2 THEN KM=7 'no f. pt needed for St & Th
3610 FOR J=1 TO KM:YT(J)=Y(J):NEXT 'install titles
3620 NT=9-NM 'no of titles
3630 GOSUB 410 'to data input
3640 GOSUB 2600 'check input data valid
3650 IF NE THEN NE=0:GOTO 3630 ELSE GOSUB 8850
3660
MW=A(2,KS):TC=A(2,KS+1):PC=A(2,KS+2):VC=A(2,KS+3):W=A(2,KS+4):TF=A(2,KS+7)+C1:
SG=A(2,KS+5):TD=A(2,KS+6)+C1 'get data in equation variables
3670 GOSUB 2150 'start calculation
3680 CLS:IF NK=0 THEN NK=1:INPUT "Temperature of estimate, deg C.":TL
3690 GOSUB 2360
3700 AV(LM)=ET:A(0,LM)=CSNG(ET) 'save answer
3710 CLS:PRINT "Viscosity of ";YN(LM);" calculated":GOSUB 7010:NM=3

```

(continued)

July

```
3720 NEXT
3730 RETURN
3740 REM Calculate visc of mixture
3750 IF AL THEN GOTO 3790
3760 IF NOT MX THEN GOSUB 3890 'conv to moles if nec
3770 GOSUB 3960 'calculate mixture visc
3780 GOTO 3860 'to return
3790 N=0:FOR A=AL TO AH STEP AD
3800 AX(1)=A:AX(2)=100-A
3810 IF NOT MX THEN GOSUB 3890
3820 GOSUB 3960
3830 IF MP THEN GOSUB 5570
3840 GOSUB 4730
3850 NEXT
3860 IF NL>2 OR AL=0 THEN GOSUB 4550 ELSE IF N THEN GOSUB 7070 'to screen if
results requ'd
3870 RETURN
3880 REM Convert to mole frn
3890 AT=0:FOR J=1 TO NL
3900 AY(J)=AX(J)/AM(J):AT=AT+AY(J)
3910 NEXT
3920 FOR J=1 TO NL
3930 AX(J)=AY(J)*100/AT
3940 NEXT:RETURN
3950 REM Visc of mixture
3960 AT=0
3970 FOR J=1 TO NL
3980 AT=AT+AX(J)*LOG(AV(J))
3990 NEXT
4000 ET=EXP(AT/100) 'viscosity of mixture
4010 RETURN
4490 REM Print results
4500 CLS
4510 PRINT "Viscosity of ";YB;" at";:PRINT USING " ###.# deg C is ###.###
cp";TL,ET
4520 PRINT
4530 RETURN
4540 REM Screen print for mixture
4550 CLS
4560 PRINT "Viscosity of mixture of:"
4570 FOR J=1 TO NL
4580 PRINT TAB(5) USING Y1;A(2,J);:PRINT " ";YU;" ";YN(J)
4590 NEXT
4600 PRINT:IF TL<>-1000 THEN PRINT "at ";TL;" deg C";
4610 PRINT " is";:PRINT USING Y3;ET;:PRINT " cp"
4620 GOSUB 7070
4630 RETURN
4640 REM Screen print list of results
4650 IF TL<>TS AND N<>0 THEN 4680
4660 N=0:PRINT "Viscosity of ";YB
4670 PRINT TAB(5) "Temperature, deg C";TAB(NS-25)"Viscosity, cp"
4680 PRINT TAB(12) USING Y2;TL;:PRINT TAB(NS-23) USING Y4;ET
4690 N=N+1
4700 IF N=NR-4 THEN GOSUB 7070:N=0:CLS
4710 RETURN
4720 REM Screen display list of mixture results
4730 IF A<>AL AND N<>0 THEN 4760 'jump titles if not first time
4740 CLS:N=0:PRINT TAB(12);YU;" of";TAB(54)"Viscosity"
4750 PRINT YN(1);TAB(25) YN(2);TAB(58)"cp"
4760 PRINT USING Y1;A;
4770 PRINT TAB(25) USING Y1;100-A;
4780 PRINT TAB(56) USING Y3;ET
4790 N=N+1
4800 IF N=NR-4 THEN GOSUB 7070:N=0
4810 RETURN
4990 REM Hardcopy results
5000 IF TN THEN CLS:PRINT "Too late! Cannot print results for range
now.":GOSUB 7070:GOTO 5150
5005 GOSUB 6000
5010 CLS:ON JP GOTO 5020,5090
5020 GOSUB 5290:GOSUB 5330
5030 LPRINT
5040 LPRINT TAB(5) "viscosity of ";YB;" estimated by method of ";YM(NM);" is:"
5050 LPRINT:LPRINT TAB(25) USING Y4+" cp at "+Y2+" deg C";ET,TL
5060 LPRINT:LPRINT
5070 GOTO 5150
```



```

5080 REM Table heading
5090 IF LT THEN 5140      'LT is flag for table
5100 GOSUB 5290
5110 LPRINT
TAB(5)"Substance";TAB(25)"Temperature";TAB(40)"Viscosity";TAB(52)"Method"
5120 LPRINT TAB(30)"deg C";TAB(44)"cp"
5130 LPRINT:LT=1
5140 LPRINT TAB(5)YB;:LPRINT TAB(29) USING Y2;TL;:LPRINT TAB(40) USING
Y4;ET;:LPRINT TAB(52) YM(NM)
5150 RETURN
5160 REM List results for temp range on printer
5170 IF LP=0 THEN 5270
5180 IF TL<>TS THEN 5250
5190 GOSUB 5290:GOSUB 5330
5200 LPRINT TAB(5)"Estimated viscosities using method of ";YM(NM)
5210 LPRINT
5220 LPRINT TAB(10)"Temperature";TAB(45)"Viscosity"
5230 LPRINT TAB(13)"deg C";TAB(49)"cp"
5240 LPRINT
5250 LPRINT TAB(14) USING Y2;TL;:LPRINT TAB(45) USING Y4;ET
5260 IF TL+TN>TE THEN LPRINT
5270 RETURN
5280 REM Print heading
5290 LPRINT TAB(5)"Project: ";YP;TAB(55)YD
5300 LPRINT
5310 RETURN
5320 REM Input data print
5330 LPRINT TAB(5)"INPUT DATA for ";YB
5340 LPRINT
5350 FOR J=1 TO 9-NM
5360 LPRINT TAB(10) Y(J);:LPRINT TAB(45) USING Y4;A(2,J)
5370 NEXT
5380 LPRINT
5390 RETURN
5400 REM Hardcopy for mixtures
5410 GOSUB 5290
5420 LPRINT TAB(40)"INPUT DATA":LPRINT
5430 LPRINT TAB(5)"Substance";TAB(35)"Visc - cp";TAB(48)"Mol. wt";TAB(63)YU
5440 FOR J=1 TO NL
5450 LPRINT TAB(5) YN(J);
5460 LPRINT TAB(35) USING Y3;AV(J);
5470 IF MC(J) THEN LPRINT"*";
5480 LPRINT TAB(48) USING Y1;AM(J);
5490 LPRINT TAB(61) USING Y1;A(2,J)
5500 NEXT
5510 LPRINT:LPRINT TAB(10) "* means estimated value, not input"
5520 LPRINT:LPRINT TAB(5) "Viscosity of mixture";
5530 IF TL<>-1000 THEN LPRINT" at";TL;" deg C";
5540 LPRINT" is";:LPRINT USING Y3;ET;:LPRINT" cp.":LPRINT
5550 RETURN
5560 REM Table of mixture viscosities
5570 IF A<>AL THEN 5700      'only print title first time
5580 GOSUB 5290
5590 LPRINT:LPRINT TAB(5) "Viscosities of mixtures of:"
5600 LPRINT:FOR J=1 TO 2
5610 LPRINT TAB(10);YN(J)
5620 NEXT
5630 LPRINT
5640 IF TL<>-1000 THEN LPRINT TAB(5)" at";TL;" deg C ";
5650 LPRINT TAB(5) "are as follows:"
5660 LPRINT:LPRINT TAB(17);YU;" of ";
5670 LPRINT TAB(55)"Viscosity"
5680 LPRINT TAB(5);YN(1);TAB(30);YN(2);TAB(58)"cp"
5690 LPRINT
5700 LPRINT TAB(5) USING Y1;A;:LPRINT TAB(30) USING Y1;100-A;
5710 LPRINT TAB(55) USING Y3;ET
5720 IF A+AD>AH THEN LPRINT
5730 RETURN
5990 REM Hardcopy option menu
6000 CLS:IF JP=2 THEN 6060
6010 PRINT@ FN LC(1,(NS-7)/2),"PRINT OPTIONS"
6020 PRINT:PRINT TAB(5)"1) Print just the current result complete with input
data"
6030 PRINT TAB(5)"2) Print table headings and start table for all":PRINT
TAB(8)"results until new project is started."

```

(continued)

```

6040 NC=2:GOSUB 7120
6050 JP=NC      'JP = flag for choice
6060 RETURN
6070 REM Main menu
6080 CLS
6090 PRINT@ FN LC(1,(NS-4)/2),"MAIN MENU"
6100 PRINT:PRINT TAB(5)"1) Viscosity of non-associating liquid - most
accurate"
6110 PRINT TAB(5)"2) Viscosity of associating liquid - not very accurate"
6120 PRINT TAB(5)"3) Viscosity of liquid mixture"
6130 NC=3:GOSUB 7120
6140 NM=NC
6150 RETURN
6160 REM Calculation options
6170 CLS
6180 PRINT@ FN LC(1,(NS-10)/2),"CALCULATION OPTIONS"
6190 PRINT:PRINT TAB(5)"1) Estimate viscosity for one specific temperature"
6200 PRINT TAB(5)"2) Estimate viscosity for a range of temperatures"
6210 PRINT
6220 NC=2:GOSUB 7120
6230 TN=0
6240 ON NC GOTO 6370
6250 CLS:INPUT"Start of temperature range, deg C";TS
6260 INPUT"End of temperature range, deg C";TE
6270 INPUT"Temperature interval in range, deg C";TN
6280 PRINT:PRINT "Everything correct, Y/N? ";
6290 GOSUB 7030
6300 IF KA>2 THEN 6250
6310 CLS:PRINT "Do you want results tabulated on printer, Y/N? "
6320 GOSUB 7030
6330 LP=(KA<3)
6340 IF LP THEN PRINT "Press <ENTER> when printer is ready" ELSE 6380
6350 Y=INKEY$:IF Y="" THEN 6350
6360 GOTO 6380
6370 CLS:INPUT"Temperature, deg C";TL
6380 RETURN
6390 REM Menu after calc
6400 PRINT TAB((NS-4)/2),"WHAT NEXT"
6410 PRINT
6420 PRINT TAB(5)"1) Hardcopy of result (one result only).\"
6430 PRINT TAB(5)"2) Viscosity for same material at another temperature.\"
6440 PRINT TAB(5)"3) Viscosity for different substance of the same type.\"
6450 PRINT TAB(5)"4) Main menu.\"
6460 PRINT TAB(5)"5) New project (all data lost).\"
6470 PRINT TAB(5)"6) End program.\"
6480 NC=6:GOSUB 7120
6490 RETURN
6500 REM Menu after mix
6510 PRINT TAB((NS-4)/2),"WHAT NEXT"
6520 PRINT
6530 PRINT TAB(5)"1) Hardcopy of results\"
6540 PRINT TAB(5)"2) Another calculation for same materials.\"
6550 PRINT TAB(5)"3) Calculate for a different mixture.\"
6560 PRINT TAB(5)"4) Main menu.\"
6570 PRINT TAB(5)"5) New project (all data lost).\"
6580 PRINT TAB(5)"6) End program.\"
6590 NC=6:GOSUB 7120
6600 RETURN
6610 REM Project intro routine
6620 CLS:INPUT"Enter project title";YP
6630 YD=LEFT$(TIME$,8):PRINT "Project date (default = ";YD;")";:INPUT YD
6640 RETURN
6990 REM Subroutines
7000 REM Delay
7010 FOR J=1 TO 1000:NEXT:RETURN
7020 REM Check inkey
7030 YR="YyNn"
7040 Y=INKEY$:IF Y="" THEN 7040
7050 KA=INSTR(YR,Y):IF KA THEN RETURN ELSE 7040
7060 REM Press key to cont
7070 PRINT@ FN LC(NR,1),"PRESS ANY KEY TO CONTINUE";
7080 Y=INKEY$:IF Y="" THEN 7080
7090 CLS
7100 RETURN
7110 REM Select from menu
7120 PRINT:PRINT"Choose by number"

```



```

7130 Y=INKEY$:IF Y="" THEN 7130
7140 Y%=VAL(Y)
7150 IF Y%<1 OR Y%>NC THEN 7130
7160 PRINT Y:NC=Y%
7170 RETURN
7180 REM Error trap
7190 CLS
7200 IF ERR/2+1=5 THEN 7250
7210 IF ERR/2+1=11 THEN 7290
7220 IF ERR/2+1=54 THEN 7360
7230 IF ERR/2+1=6 THEN 7270
7240 PRINT "Error in line";ERL:ON ERROR GOTO 0:RESUME
7250 PRINT "Illegal function call error has occurred in line";ERL:GOSUB 7340
7260 PRINT "Look for negative or zero values of input data":GOTO 7310
7270 PRINT "Overflow error has occurred in line";ERL:GOSUB 7340
7280 PRINT "Look for very low or very high values of input data":GOTO 7310
7290 PRINT "Division by zero error has occurred in line";ERL:GOSUB 7340
7300 PRINT "Look for very low or zero values of input data"
7310 PRINT "Check that input data units are correct, and that all values
are within the range of the correlation"
7320 GOSUB 7070:NE=1 'ne is error flag
7330 RESUME 1030
7340 PRINT "Program will return to data entry to allow you to check validity of
your input data"
7350 RETURN
7360 PRINT "No such file"
7370 GOSUB 7070:CLOSE:JF=1:RESUME NEXT
7990 REM Introductory blurb
8000 CLS
8010 GOSUB 8220:IF NB THEN 8200
8020 CLS:PRINT "This program estimates viscosities of pure liquids and
mixtures"
8030 PRINT:PRINT "The viscosities of pure, non-associating, liquids are
estimated by the method of J.W.Przedziecki and T.Sridhar published in the
AIChEJ, Feb. 1985, p333. It is the most accurate method and should be used if
possible."
8040 PRINT:PRINT "Data required for the estimate are the molecular weight,
the critical properties, the acentric factor, and at least one value of density
at a known temperature."
8050 PRINT:PRINT "This method gives estimates with an average error of 9%, and
a maximum error of 40%, but results are very unreliable if used for associating
liquids."
8060 PRINT:GOSUB 7070
8070 CLS
8080 PRINT "For associating liquids there is really no reliable
predictive method."
8090 PRINT:PRINT "The method used in this program is that of Stiel and
Thodos, as described in 'The Properties of Gases and Liquids' by Reid and
Sherwood, 2nd Ed., p437."
8100 PRINT:PRINT "Data required for the estimate are the molecular weight,
the critical properties, the acentric factor, and at least one value of density
at a known temperature."
8110 PRINT:PRINT "This method gives quite large errors (as much as -
80%), usually on the low side, for some associating liquids, but no other method
is any more accurate."
8120 PRINT:GOSUB 7070
8130 CLS
8140 PRINT "Prediction of viscosities for liquid mixtures from pure component
data is also rather unreliable."
8150 PRINT:PRINT "The method used in this program is procedure 8H from the
AIChE 'Data Prediction Manual'. It is appropriate for systems where the
mixtures have viscosities intermediate between those of the components."
8160 PRINT:PRINT "Data required for the estimate are either the viscosities
of the components, or the data listed previously to allow them to be
estimated.";
8170 PRINT "Liquid compositions may be entered in mol% or wt%, but in the
latter case the molecular weight is also required."
8180 PRINT:PRINT "This method is not suitable for systems that have maxima
or minima in the mixture properties."
8190 GOSUB 7070
8200 RETURN
8210 REM Routine to bypass blurbs
8220 PRINT "This program includes explanatory text and instructions. If you are
familiar with the program, you can bypass these. Do you want instructions
displayed for this run, Y/N?"

```

(continued)


```

8230 GOSUB 7030
8240 NB=(KA>2) 'nb is flag indicating No Blurbs
8250 RETURN
8260 REM Blurb for P&S method
8270 CLS:IF NB THEN 8450
8280 IF NM=2 THEN GOTO 8370
8290 PRINT "This calculation method gives quite accurate results for most"
8300 PRINT "non-associating compounds. Any errors tend to be on the low"
8310 PRINT "side, and the errors are more pronounced at lower temperatures,"
8320 PRINT "reaching a maximum close to the freezing point."
8330 PRINT:PRINT "The only common organics that cannot be handled are
alcohols,"
8340 PRINT "both aliphatic and aromatic. Cyclic and branched compounds
":PRINT"tend to give quite large errors, usually on the low side."
8350 PRINT "Organic acids seem to be predictable."
8360 GOTO 8440
8370 CLS
8380 PRINT "This calculation method gives reasonable results for many"
8390 PRINT "compounds. It works best at temperatures from"
8400 PRINT "the boiling point up, and the errors are more pronounced":PRINT
"at lower temperatures."
8410 PRINT "Only use this method for associating compounds."
8420 PRINT:PRINT "For all other substances, the Przedziecki and Sridhar"
8430 PRINT "method is better."
8440 GOSUB 7070
8450 RETURN
8460 REM Blurb for mixtures
8470 IF NB THEN 8530
8480 CLS
8490 PRINT "This section calculates the viscosity of liquid mixtures from the
viscosities of the pure components, for up to 10 components."
8500 PRINT:PRINT "If the pure component viscosities are not known, they will
be estimated by one of the methods available. Leaving the viscosity value as
zero during data entry will automatically activate the estimating procedure."
8510 PRINT:PRINT "If there are only two components, a range of mixtures can
be calculated automatically - you will be prompted for the range if this option
is chosen."
8520 GOSUB 7070
8530 RETURN
8540 REM Read data from file
8550 CLS
8560 PRINT "Checking disk for data..."
8570 GOSUB 8780:GOSUB 8670:GOSUB 8720
8580 IF N>LOF(1)/LF THEN 8650
8590 A(2,KS)=CVS(MF$):A(2,KS+1)=CVS(TF$)
8600 A(2,KS+2)=CVS(PF$):A(2,KS+3)=CVS(VF$)
8610 A(2,KS+4)=CVS(WF$):A(2,KS+5)=CVS(DF$)
8620 A(2,KS+6)=CVS(SF$):A(2,KS+7)=CVS(FF$)
8630 FOR J=1 TO 7:IF ABS(A(2,KS+J))<C3 THEN A(2,KS+J)=0
8640 NEXT
8650 RETURN
8660 REM Open file for crit data
8670 Y=LEFT$(YH,1)+"CRIT/DAT"
8680 OPEN "R",1,Y,128
8690 FIELD 1,32 AS NF$,4 AS MF$,4 AS BF$,4 AS TF$,4 AS PF$,4 AS VF$,4 AS WF$,4
AS DF$,4 AS SF$,4 AS FF$,60 AS XF$
8700 RETURN
8710 REM Find if name in record
8720 N=1
8730 GET 1,N
8740 J=LEN(YH):K=LEN(NF$):IF LEFT$(NF$,J)=YH THEN IF RIGHT$(NF$,K-
J)=STRING$(K-J," ") THEN 8760
8750 N=N+1:IF N<=LOF(1)/LF THEN 8730
8760 RETURN
8770 REM Convert name to caps
8780 YH=YB
8790 FOR J=1 TO LEN(YB)
8800 K=ASC(MID$(YB,J))
8810 IF K>96 AND K<123 THEN YL=CHR$(K-32):MID$(YH,J)=YL
8820 NEXT
8830 RETURN
8840 REM Write data to disk
8850 IF N<=LOF(1)/LF THEN LSET FF$=MKS$(A(2,KS+7)):GOTO 8920
8860 LSET NF$=YH

```



```

8870 LSET MF$=MKS$(A(2,KS)):LSET TF$=MKS$(A(2,KS+1))
8880 LSET PF$=MKS$(A(2,KS+2)):LSET VF$=MKS$(A(2,KS+3))
8890 LSET WF$=MKS$(A(2,KS+4)):LSET DF$=MKS$(A(2,KS+5))
8900 LSET SF$=MKS$(A(2,KS+6)):LSET FF$=MKS$(A(2,KS+7))
8910 LSET BF$=" "
8920 PUT 1,N
8930 CLOSE 1
8940 RETURN

```

critical.bas

"Small-Scale Engineering Applications," by J. Neil Stone. July, page 253.

```

10 REM Predict critical props program - Apr 85 - CRITICAL/BAS
20 REM Rummens & Rajan, Can.J.Ch.E., Jun 1979, 349 - general
30 REM Edited for BYTE Dec 1985
40 GOTO 170
50 DATA"***** CRITICAL PROPERTY *****"
60 DATA"*** PREDICTION ***"
70 DATA" by"
80 DATA" J.Neil Stone"
90 DATA" Ledge Engineering Inc"
100 DATA" 179 Lansdowne Avenue"
110 DATA" Kingsville, Ontario, N9Y 3J2"
120 DATA" "
130 DATA" **** Apr 1985 ****"
140 CLS:FOR J=1 TO 9:READ Y:PRINT@ FN LC(NR/2-5+J,NS/2-17),Y:NEXT
150 RETURN
160 REM Initialize
170 CLEAR 500
180 DEFINT J-L,N:DEFSTR R,X-Z
190 DIM Y,K,J,M,TD,Y%,LF,KA
200 POKE 16400,1 'Set caps mode
210 NS=64:NR=16:LF=1 'parameters for TRS80
220 'NS=80:NR=24:LF=128 'parameters for 80x24
230 DEF FN LC(J,K)=(J-1)*NS+K-1
240 GOSUB 50 'display title
250 DIM PB(3),E(2,11),Y(12),P(2),TR(2),F(1),V(2) 'arrays
260 REM Set up message strings
270 Y(1)="Molecular weight":Y(5)="B.Pt.":Y(2)=Y(5)+" deg C":Y(3)="Specific
gravity, 20/4":Y(4)=" (opt)":Y(6)="Vap.Press. ":ZZ(8)="Density,
g/cc":Y(8)="deg F":Y(9)="deg C":Y(10)="mmHg":Y(11)="psia":Y(12)="kPa"
280 Y1="####.##":Y2="####.####":Y3="####.###"
290 Z="-->":Z1=CHR$(10)+CHR$(91)+CHR$(9)+CHR$(13)+"X"+"x":Z3=" ":Z4=".-
0123456789":Z2=Z4+Z1:Z5=CHR$(8)+CHR$(13)+"ED"+Z4
300 Z8="USE UP, DN, RT ARROWS TO SELECT ENTRY. <X> TO EXIT":Z9="USE <-- TO
CORRECT. PRESS <ENTER> WHEN DONE":ZA=STRING$(17," ")
310 REM Set up constants
320 C1=1.5:C2=36:C3=42:C4=35:C5=.3143:C6=.0838:C7=8.315E-
03:C8=1.9:C9=.26:CA=2.38:CB=.2:CC=.489:CD=.225:CE=.511:CF=660:CG=.1:CH=1.8:CJ=
32:CK=.10135:CL=760:CM=14.696:CN=1000:CP=273.16
330 ON ERROR GOTO 6130
340 GOTO 1500 'to main program
490 REM Screen routine - put titles in ZZ()
500 CLS:PRINT@ FN LC(1,(NS-LEN(RA))/2),RA
510 PRINT STRING$(NS-1," ")
520 FOR J=KL TO KM
530 IF J=7 THEN PRINT:GOTO 570
540 IFJ=KL THEN PRINT TAB(5) ZZ(J) TAB(NS-14) E(2,J):GOTO 570
550 PRINT TAB(5) ZZ(J);
560 IF J>3 THEN PRINT TAB(NS-30) E(1,J) TAB(NS-14) E(2,J) ELSE PRINT
570 NEXT
580 PRINT@ FN LC(NR-3,1),STRING$(NS-1,95)
590 PRINT@ FN LC(NR-2,1),CHR$(30);RB;
600 KQ=NS-17:KP=KL+1:J9=3:GOTO 660
610 PRINT@ FN LC(KP+1,KQ+1),Z;
620 PRINT@ FN LC(NR-1,1),CHR$(30);Z8;
630 Y$=INKEY$:IF Y$="" THEN 630
640 IF INSTR(Z2,Y$)=0 THEN 630
650 J9=INSTR(Z1,Y$)
660 IF J9=0 THEN 800 ELSE PRINT@ FN LC(KP+1,KQ+1),Z3;:ON J9 GOTO

```

(continued)

```

690,680,710,680
670 RETURN
680 J9=J9-3
690 KP=KP+J9:IF KP>KM+1 THEN KP=KL+1 ELSE IF KP<KL+1 THEN KP=KM+1
700 GOTO 750
710 ON J8 GOTO 750,740
720 IF KQ=NS-49 OR KQ=NS-33 THEN KQ=KQ+16 ELSE KQ=NS-49
730 GOTO 750
740 IF KQ=NS-33 THEN KQ=NS-17 ELSE KQ=NS-33
750 ZF=STR$(KP-1):ZF=RIGHT$(ZF,LEN(ZF)-1)+", "
760 IF INSTR(RD,ZF) THEN IF J9=3 THEN J9=1:GOTO 690 ELSE 690
770 IF INSTR(RE,ZF) THEN KQ=NS-17
780 GOTO 610
790 REM Data entry section
800 PRINT@ FN LC(NR-1,1),CHR$(30);Z9;
810 KC=1:ZE=Y$
820 PRINT@ FN LC(KP+1,KQ+1),ZA;:PRINT@ FN LC(KP+1,KQ+1)+4,ZE;CHR$(95);
830 Y$=INKEY$:IF Y$="" THEN 830
840 IF INSTR(Z5,Y$)=0 THEN 830
850 ON INSTR(Z5,Y$) GOTO 900,920
860 KC=KC+1
870 ZE=ZE+Y$
880 PRINT@ FN LC(KP+1,KQ+1)+KC+3,Y$;CHR$(95);
890 GOTO 830
900 IF KC=0 THEN 830 ELSE KC=KC-1:IF KC=0 THEN ZE="" ELSE ZE=LEFT$(ZE,KC)
910 GOTO 820
920 KE=(KQ-NS+49)/16
930 E(KE,KP-1)=VAL(ZE)
940 PRINT@ FN LC(KP+1,KQ+1),ZA;:PRINT@ FN LC(KP+1,KQ+1)+3,CSNG(VAL(ZE));
950 ZE="":J9=1
960 IF KQ=NS-17 THEN KP=KP+1
970 IF KP>KM+1 THEN KP=KL+1
980 GOTO 710
1490 REM Main program
1500 GOSUB 6500 'introductory text
1510 GOSUB 4610 'get title etc
1520 GOSUB 3000 'get initial info
1530 GOSUB 2000 'input data
1540 GOSUB 2500 'calculate critprops
1550 GOSUB 3500 'results on screen
1560 GOSUB 4500 'What next?
1570 ON NC GOTO 1590,1600,1610,1610
1580 CLS:ON ERROR GOTO 0:END
1590 GOSUB 4000:GOTO 1560 'to hardcopy routine
1600 GOSUB 5000:GOTO 1560 'save to disk
1610 FOR J=1 TO 2:FOR K=1 TO 8:E(J,K)=0:NEXTK,J 'clean out input array
1620 ON NC-2 GOTO 1510,1520
1990 REM Data input
2000 CLS
2010 RA="INPUT DATA FOR "+YA
2020 RB="DATA FOR SECOND VAPOUR PRESSURE POINT OPTIONAL"
2030 RD="2,3,7,"
2040 RE="1,"
2050 J8=2
2060 KL=1:KM=8:ON JF GOTO 2080,2080,2090
2070 E(1,4)=CK*CN:GOTO 2100
2080 E(1,4)=CL:GOTO 2100
2090 E(1,4)=CM
2100 ZZ(1)=Y(1):ZZ(2)=STRING$(NS-36," ")+"Value"+STRING$(9,"
")+"Temperature":ZZ(3)=STRING$(43," ")+"YT:ZZ(4)=Y(5)+
"+YP+"/"YT:ZZ(5)=Y(6)+" "+YP:ZZ(6)=ZZ(5):ZZ(7)=" "
2110 GOSUB 500 'to screen entry
2120 M=E(2,1)
2130 N=4
2140 FOR J=0 TO 2
2150 ON JF GOTO 2170,2180,2190
2160 P(J)=E(1,J+N)/CN:T(J)=E(2,J+N)+CP:C=CN*CK:GOTO 2200
2170 P(J)=E(1,J+N)*CK/CL:T(J)=E(2,J+N)+CP:C=CL:GOTO 2200
2180 P(J)=E(1,J+N)*CK/CL:T(J)=(E(2,J+N)-CJ)/CH+CP:C=CL:GOTO 2200
2190 P(J)=E(1,J+N)*CK/CM:T(J)=(E(2,J+N)-CJ)/CH+CP:C=CM
2200 NEXT
2210 SG=E(1,8):D=SG*CN
2220 IF JF=2 OR JF=3 THEN T=(E(2,8)-CJ)/CH ELSE T=E(2,8)
2230 T(3)=T+CP
2240 RETURN
2490 REM Calculation

```



```

2500 CLS
2510 PRINT"Calculating"
2520 IF P(2)=0 THEN N=0 ELSE N=1
2530 IF P(0)=0 THEN GOSUB 2710 ELSE TB=T(0):TD=T(0)
2540 T1=C1*TD
2550 FOR J=0 TO 1
2560 TR(J)=T(N+J)/T1
2570 F(J)=C2/TR(J)+C3*LOG(TR(J))-C4-TR(J)^6
2580 NEXT
2590 DF=F(0)-F(1)
2600 AL=(LOG(P(N)/P(N+1))-C5*DF)/(LOG(T(N)/T(N+1))-C6*DF)
2610 PC=EXP(LOG(P(N))-C5*F(0)-AL*(LOG(TR(0))-C6*F(0)))
2620 VC=C7*T1/PC/(C8+C9*AL)
2630 V(1)=VC/(CA+CB*AL)
2640 TR(2)=T(3)/T1:V(2)=M*(CC-CD*TR(2)+CE*((1-TR(2))^(1/3)))/D
2650 T2=T1+CF*(V(1)-V(2))
2660 IF ABS(T2-T1)<=CG THEN 2680
2670 T1=T2:GOTO 2550 'loop if not converged
2680 PB(0)=T1:PB(1)=PC:PB(2)=VC*CN:PB(3)=.2033*AL-1.1816
2690 RETURN
2700 REM Estimate bpt by Antoine
2710 B=LOG(P(1)/P(2))/(1/T(2)-1/T(1))
2720 A=LOG(P(1))+B/T(1)
2730 TD=B/(A-LOG(C))
2740 RETURN
2980 REM
2990 REM Compound name
3000 INPUT"Name of compound";YA
3010 IF JF THEN 3130 'jf is unit flag
3020 PRINT:PRINT"Which units are you using?"
3030 PRINT:PRINT"1) mmHg and deg C"
3040 PRINT"2) mmHg and deg F"
3050 PRINT"3) psia and deg F"
3060 PRINT"4) kPa and deg C"
3070 NC=4:GOSUB 6070
3080 JF=NC:ON JF GOTO 3100,3110,3120
3090 YP=Y(12):YT=Y(9):GOTO 3130
3100 YP=Y(10):YT=Y(9):GOTO 3130
3110 YP=Y(10):YT=Y(8):GOTO 3130
3120 YP=Y(11):YT=Y(8)
3130 RETURN
3490 REM Display results on screen
3500 CLS
3510 YC="CRITICAL PROPERTIES OF "+YA:J=(NS-LEN(YC))/2
3520 PRINT TAB(J);YC
3530 PRINT
3540 J=(NS-40)/2:K=(NS+30)/2
3550 PRINT TAB(J)"Critical temperature, K:":PRINT TAB(K) USING Y1;PB(0)
3560 PRINT TAB(J)"Critical pressure, MPa:":PRINT TAB(K) USING Y3;PB(1)
3570 PRINT TAB(J+19)"atma:":PRINT TAB(K) USING Y1;PB(1)/CK
3580 PRINT TAB(J)"Critical volume, cc/mol:":PRINT TAB(K) USING Y1;PB(2)
3590 PRINT
3600 PRINT TAB(J)"Pitzer acentric factor :":PRINT TAB(K) USING Y2;PB(3)
3610 PRINT TAB(J)"Riedel critical parameter :":PRINT TAB(K) USING Y2;AL
3620 GOSUB 6030
3630 RETURN
3990 REM printer output
4000 CLS
4010 J=PEEK(14312) AND 240 'check if printer ready
4020 IF J=48 THEN 4060
4030 PRINT"Printer not on line - press <ENTER> when ready"
4040 Y=INKEY$:IF Y="" THEN 4040
4050 GOTO 4000
4060 IF LP THEN 4370
4070 PRINT"If you are going to calculate properties for several
substances":PRINT"the results can be printed as a table."
4080 PRINT"Otherwise they will be printed with each set as a
separate":PRINT"report."
4090 PRINT"Press <T> for tabulated results, <S> for separate reports";
4100 YR="TtSe":GOSUB 6010
4110 LP=(KA<3)
4120 LPRINT TAB(10)"Project: ";YB;TAB(70)YD:LPRINT
4130 IF LP THEN 4270
4140 REM Single printout
4150 J=(80-LEN(YC))/2:LPRINT TAB(J);YC

```

(continued)


```

4160 LPRINT
4170 LPRINT TAB(18)"Critical temperature, K:":LPRINT TAB(55) USING Y1;PB(0)
4180 LPRINT TAB(18)"Critical pressure, MPa:":LPRINT TAB(55) USING Y3;PB(1)
4190 LPRINT TAB(37)"atma:":LPRINT TAB(55) USING Y1;PB(1)/.10133
4200 LPRINT TAB(18)"Critical volume, cc/mol:":LPRINT TAB(55) USING Y1;PB(2)
4210 LPRINT
4220 LPRINT TAB(18)"Pitzer acentric factor :":LPRINT TAB(55) USING Y2;PB(3)
4230 LPRINT TAB(18)"Riedel critical parameter :":LPRINT TAB(55) USING
Y2;AL:GOTO 4240
4240 LPRINT:LPRINT TAB(10)"These properties calculated using the method of
Rummen and Rajan":LPRINT TAB(10)"with the following input data:":LPRINT
4250 LPRINT TAB(18)"Molecular weight":LPRINT TAB(55) USING Y3;M
4260 IF P(0) THEN LPRINT TAB(18) Y(5)+" "+YT:LPRINT TAB(55) USING Y1;E(2,4)
4270 LPRINT TAB(18) ZZ(5):LPRINT TAB(45) USING Y2+" "+YP+" @"+Y1+"
"+YT;E(1,5),E(2,5)
4280 IF E(1,6) THEN LPRINT TAB(18) ZZ(6):LPRINT TAB(45) USING Y2+" "+YP+"
@"+Y1+" "+YT;E(1,6),E(2,6)
4290 LPRINT TAB(18)ZZ(8):LPRINT TAB(45) USING Y2+" g/cc"+" @"+Y1+"
"+YT;E(1,8),E(2,8)
4300 LPRINTCHR$(12)
4310 GOTO 4380
4320 REM Tabular printout
4330 LPRINT TAB(10)"Name";TAB(50)"Critical";TAB(70)"Acentric"
4340 LPRINT TAB(40)"Temp";TAB(50)"Pressure";TAB(62)"Volume";TAB(70)"factor"
4350 LPRINT TAB(40)" K";TAB(49)"MPa atm";TAB(62)"cc/mol"
4360 LPRINT
4370 LPRINT TAB(10);YA:LPRINT TAB(40) USING Y1;PB(0):LPRINT TAB(47) USING
Y3;PB(1):LPRINT TAB(55) USING Y1;PB(1)/.10133:LPRINT TAB(62) USING
Y1;PB(2):LPRINT TAB(70) USING Y2;PB(3)
4380 RETURN
4490 REM Menu
4500 CLS
4510 PRINT@ FN LC(1,NS/2-3),"WHAT NEXT"
4520 PRINT:PRINT TAB(10)"1) Hardcopy of results"
4530 PRINT TAB(10)"2) Save critical properties to disk"
4540 PRINT TAB(10)"3) New project"
4550 PRINT TAB(10)"4) Calculate another material"
4560 PRINT TAB(10)"5) End program"
4570 NC=5:GOSUB 6070
4580 RETURN
4600 REM Project Intro routine
4610 CLS:LP=0:JF=0:INPUT"Enter project title":YB
4620 YD=LEFT$(TIME$,8):PRINT"Project date (default = ";YD;")":INPUT YD
4630 RETURN
4990 REM Save critical data
5000 GOSUB 5240:GOSUB 5120:GOSUB 5170
5010 CLS:PRINT"Saving to file ";Y
5020 IF N>LOF(1)/LF THEN LSET XF$=STRING$(64," ")
5030 LSETNF$=YH
5040 LSETMF$=MKS$(M):LSETBF$=MKS$(TB)
5050 LSETTF$=MKS$(PB(0)):LSETPF$=MKS$(PB(1)/.10133)
5060 LSETVF$=MKS$(PB(2)):LSETWF$=MKS$(PB(3))
5070 LSETDF$=MKS$(SG):LSETSF$=MKS$(T)
5080 PUT1,N
5090 CLOSE 1
5100 RETURN
5110 REM Open file for crit data
5120 Y=LEFT$(YA,1)+"CRIT/DAT"
5130 OPEN"R",1,Y,128
5140 FIELD 1,32 AS NF$,4 AS MF$,4 AS BF$,4 AS TF$,4 AS PF$,4 AS VF$,4 AS WF$,4
AS DF$,4 AS SF$,64 AS XF$
5150 RETURN
5160 REM Find if name in record
5170 N=1
5180 IF N>LOF(1)/LF THEN 5220
5190 GET 1,N
5200 J=LEN(YH):K=LEN(NF$):IF LEFT$(NF$,J)=YH THEN IF RIGHT$(NF$,K-
J)=STRING$(K-J," ") THEN 5220
5210 N=N+1:GOTO 5180
5220 RETURN
5230 REM Convert name to caps
5240 YH=YA
5250 FOR J=1 TO LEN(YA)
5260 K=ASC(MID$(YA,J))
5270 IF K>96 AND K<123 THEN YL=CHR$(K-32):MID$(YH,J)=YL
5280 NEXT

```



```

5290 RETURN
5990 REM Subroutines
6000 YR="YyNn"
6010 Y=INKEY$:IF Y="" THEN 6010
6020 KA=INSTR(YR,Y):IF KA THEN PRINT " ";Y:RETURN ELSE 6010
6030 PRINT@ FN LC(NR-1,1),"PRESS ANY KEY TO CONTINUE";
6040 Y=INKEY$:IF Y="" THEN 6040
6050 CLS
6060 RETURN
6070 PRINT:PRINT"Choose by number ";
6080 Y=INKEY$:IF Y="" THEN 6080
6090 Y%=VAL(Y)
6100 IF Y%<1 OR Y%>NC THEN 6080
6110 PRINT Y:NC=Y%
6120 RETURN
6130 CLS
6140 IF ERR/2+1=5 OR ERR/2+1=11 OR ERR/2+1=6 THEN 6160
6150 PRINT"Error in line";ERL:ON ERROR GOTO 0:RESUME
6160 PRINT"Math error has occurred in line";ERL
6170 PRINT"Program will return to data entry to allow you to re-enter your
input data."
6180 PRINT"Look for very low or very high, zero or negative values
of":PRINT"input data."
6190 PRINT"Check that input data units are correct, and that all values
are within the range of the correlation."
6200 GOSUB 6030: RESUME 1530
6490 REM Opening text
6500 CLS
6510 PRINT"This program calculates the critical properties and
acentric":PRINT"factors for many compounds.":PRINT
6600 PRINT"It is based on the correlation by Rummens and Rajan,
published":PRINT"in the Can.J.Ch.E., Jun 1979, (Vol 57, No.3), page 349."
6610 PRINT:PRINT"Input data required are the molecular weight, the vapor"
6620 PRINT"pressure at any two temperatures, and the density at
any":PRINT"temperature."
6630 PRINT:PRINT"One of the vapor pressure points may be the normal
boiling":PRINT"point, although the authors recommend against this,
and":PRINT"suggest two other points 40 to 60 K apart will give best results"
6640 GOSUB 6030
6650 RETURN

```

listing1.bas

"Small-Scale Engineering Applications," by J. Neil
Stone. July, page 253.

```

8540 REM Read data from file
8550 CLS
8560 PRINT"Checking disk for data..."
8570 GOSUB 8780:GOSUB 8670:GOSUB 8720
8580 IF N>LOF(1)/LF THEN 8650
8590 A(2,KS)=CVS(MF$):A(2,KS+1)=CVS(TF$)
8600 A(2,KS+2)=CVS(PF$):A(2,KS+3)=CVS(VF$)
8610 A(2,KS+4)=CVS(WF$):A(2,KS+5)=CVS(DF$)
8620 A(2,KS+6)=CVS(SF$):A(2,KS+7)=CVS(FF$)
8630 FOR J=1 TO 7:IF ABS(A(2,KS+J))<C3 THEN A(2,KS+J)=0
8640 NEXT
8650 RETURN
8660 REM Open file for crit data
8670 Y=LEFT$(YH,1)+"CRIT/DAT"
8680 OPEN"R",1,Y,128
8690 FIELD 1,32 AS NF$,4 AS MF$,4 AS BF$,4 AS TF$,4 AS PF$,
4 AS VF$,4 AS WF$,4 AS DF$,4 AS SF$,60 AS FF$,60 AS XF$
8700 RETURN
8710 REM Find if name in record
8720 N=1
8730 GET 1,N
8740 J=LEN(YH):K=LEN(NF$):IF LEFT$(NF$,J)=YH THEN IF
RIGHT$(NF$,K-J)=STRING$(K-J," ") THEN 8760
8750 N=N+1:IF N<=LOF(1)/LF THEN 8730
8760 RETURN
8770 REM Convert name to caps

```

(continued)

```

8780 YH=YB
8790 FOR J=1 TO LEN(YB)
8800 K=ASC(MID$(YB,J))
8810 IF K>96 AND K<123 THEN YL=CHR$(K-32):MID$(YH,J)=YL
8820 NEXT
8830 RETURN
8840 REM Write data to disk
8850 IF N=LOF(1)/LF THEN LSET FF$=MKS$(A(2,KS+7)):
      GOTO 8920
8860 LSET NF$=YH
8870 LSET MF$=MKS$(A(2,KS)):LSET TF$=MKS$(A(2,KS+1))
8880 LSET PF$=MKS$(A(2,KS+2)):LSET VF$=MKS$(A(2,KS+3))
8890 LSET WF$=MKS$(A(2,KS+4)):LSET DF$=MKS$(A(2,KS+5))
8900 LSET SF$=MKS$(A(2,KS+6)):LSET FF$=MKS$(A(2,KS+7))
8910 LSET BF$=" "
8920 PUT 1,N
8930 CLOSE 1
8940 RETURN

```

listing2.bas

"Small-Scale Engineering Applications," by J. Neil Stone. July, page 253.

```

200 NS=64:NR=16:LF=1 'screen parameters
for TRS80
210 'NS=80:NR=24:LF=128 'screen parameters
for 80x24 (IBM)
220 DEF FN LC(J,K)=(J-1)*NS+K-1

```

viscibm.bas

"Small-Scale Engineering Applications," by J. Neil Stone. July, page 253.

```

0 REM ESTIMATE VISCOSITY - May 85 - VISCIBM.BAS. Version for BASICA
10 REM Final, Sept 2nd, 1985
20 REM Data input screen based on concept of L.E.Sparks, ACCESS 1982, p10
30 REM Edited for BYTE Dec 85
40 GOTO 160
50 DATA"***** VISCOSITIES *****"
60 DATA"* OF LIQUIDS AND MIXTURES *"
70 DATA" by"
80 DATA" J.Neil Stone"
90 DATA" Ledge Engineering Inc"
100 DATA" 179 Lansdowne Avenue"
110 DATA" Kingsville, Ontario, N9Y 3J2"
120 DATA" "
130 DATA" **** May 1985 ****"
140 CLS:FOR J=1 TO 9:READ Y:LOCATE NR/2-5+J,NS/2-17:PRINT Y:NEXT
150 RETURN
160 CLEAR 1500
170 DEFINT J-L,N:DEFSTR X-Z
180 DIM
Y,J,K,NS,KP,Y%,LC,KQ,AT,KC,N,UT,A,J9,ET,TR,TL,J8,J$,KS,RD,NL,NC,F2,NR,V,NM
'simple vars
210 NS=80:NR=24:LF=128 ' parameters for 80x24 (IBM)
220 DEF FN LC(J,K)=(J-1)*NS+K-1
230 GOSUB 50 'display title
240 DIM A(2,92),AX(10),AY(10),YT(8),Y(8),AV(10),AM(10),YN(10),YM(3),MC(10)
'arrays
250 REM Set up message strings
260 Y(1)="Molecular weight":Y(2)="Critical temp, K":Y(3)="Critical press,
atm":Y(4)="Critical vol, cc/mol":Y(5)="Acentric factor":Y(8)="Melting point,
deg C":Y(6)="Density, g/cc (SG)":Y(7)="Temp. of density measure, deg C"
270 YM(1)="Przedziecki & Sridhar":YM(2)="Stiel & Thodos":YM(3)="AIChE Data
Prediction Manual"
280 Z="-->":Z1=CHR$(91)+CHR$(10)+CHR$(13)+"X"+CHR$(9):Z3=" " :Z4="<--"
":Z6=".-0123456789":Z2=Z6+Z1:Z5=Z6+"ED"+CHR$(13)+CHR$(8)
290 Z8="USE UP, DOWN, RT ARROWS TO SELECT ENTRY. <X> TO EXIT.":Z9="USE "+Z4+"

```



```

TO CORRECT. PRESS <ENTER> WHEN DONE":ZC=STRING$(17," "):ZS=STRING$(79," ")
300 Y1="###.##":Y2="###":Y3="###.###":Y4="###.###":Y5="###.###^^^"
310 REM Set up constants
320 C1=273.16:C2=2/7:C3=1E-10
330 ON ERROR GOTO 7190
340 GOTO 1000 'to main prog
390 REM All-purpose screen for 1,2 or 3 col numeric entry
400 REM Screen layout
410 CLS:PRINT TAB((NS-LEN(ZA))/2) ZA
420 PRINT STRING$(NS-1,"=")
430 FOR J=1 TO NT:J$=RIGHT$(STR$(J),1)
440 PRINT YT(J);
450 IF INSTR(ZD,J$) THEN PRINT:GOTO 490 ELSE IF INSTR(ZG,J$) THEN 480
460 IF J8>2 THEN PRINT TAB(NS-45);A(0,KS+J-1);
470 IF J8>1 THEN PRINT TAB(NS-29);A(1,KS+J-1);
480 PRINT TAB(NS-13);A(2,KS+J-1)
490 NEXT
500 REM Display data as SP to prevent overflows
510 LOCATE 14,1:PRINT STRING$(NS-1,95)
520 LOCATE 15,1:PRINT ZS;:LOCATE 15,1:PRINT ZB;
530 REM Begin display
540 KQ=NS-17:KP=KL+1:J9=6:GOTO 600
550 LOCATE KP+1,KQ+1:PRINT Z;
560 LOCATE 16,1:PRINT ZS;:LOCATE 16,1:PRINT Z8;
570 Y$=INKEY$:IF Y$="" THEN 570
580 IF LEN(Y)=1 THEN IF INSTR(Z2,Y$)=0 THEN 570
590 IF LEN(Y)=1 THEN J9=INSTR(Z1,Y$) ELSE J9=ASC(RIGHT$(Y,1)):IF J9=80 THEN
J9=2 ELSE IF J9=72 THEN J9=1 ELSE 570
600 IF J9=0 THEN 760 ELSE LOCATE KP+1,KQ+1:PRINT Z3;
610 ON J9 GOTO 640,620,620,630,630,670
620 J9=1:GOTO 650
630 RETURN
640 J9=-1
650 KP=KP+J9:IF KP>KM+1 THEN KP=KL+1 ELSE IF KP<KL+1 THEN KP=KM+1
660 GOTO 710
670 ON J8 GOTO 710,700
680 IF KQ=NS-49 OR KQ=NS-33 THEN KQ=KQ+16 ELSE KQ=NS-49
690 GOTO 710
700 IF KQ=NS-33 THEN KQ=NS-17 ELSE KQ=NS-33
710 ZF=STR$(KP-1):ZF=RIGHT$(ZF,LEN(ZF)-1)+", "
720 IF INSTR(ZD,ZF) THEN IF J9=6 THEN J9=1:GOTO 650 ELSE 650
730 IF INSTR(ZG,ZF) THEN KQ=NS-17
740 GOTO 550
750 REM Data entry section
760 LOCATE 16,1:PRINT ZS;:LOCATE 16,1:PRINT Z9;
770 KC=1:ZE=Y$
780 LOCATE KP+1,KQ+1:PRINT ZC;:LOCATE KP+1,KQ+5:PRINT ZE;CHR$(95);
790 Y$=INKEY$:IF Y$="" THEN 790
800 Y%=INSTR(Z5,Y$):IF Y%=0 THEN IF LEN(Y$)>1 AND ASC(RIGHT$(Y$,1))=75 THEN
860 ELSE 790
810 IF Y%>15 THEN ON Y%-15 GOTO 880,860
820 KC=KC+1
830 ZE=ZE+Y$
840 LOCATE KP+1,KQ+KC+4:PRINT Y$;CHR$(95);
850 GOTO 790
860 KC=KC-1:IF KC<=0 THEN KC=0:ZE="" ELSE ZE=LEFT$(ZE,KC)
870 GOTO 780
880 KE=(KQ-NS+49)/16
890 A(KE,KP+KS-2)=VAL(ZE)
900 LOCATE KP+1,KQ+1:PRINT ZC;:LOCATE KP+1,KQ+4:PRINT CSNG(VAL(ZE));
910 ZE="":J9=1
920 IF KQ=NS-17 THEN KP=KP+1
930 IF KP>KM+1 THEN KP=KL+1
940 GOTO 670
990 REM Main program
1000 GOSUB 8000 'introductory text
1010 GOSUB 6620 'title etc
1020 GOSUB 6080 'main menu
1030 ON NM GOSUB 1250,1250,1500 'goto correct routine
1040 IF NC<>6 THEN 1060
1050 ON ERROR GOTO 0:END 'end of program
1060 FOR K=0 TO 2:FOR J=1 TO 8:A(K,J)=0:NEXTJ,K:LP=0
1070 IF NC=5 THEN JP=0:LT=0
1080 ON NC-3 GOTO 1020,1010
1240 REM Viscosity prediction by both methods

```

(continued)

```

1250 IF NE THEN 1290 'go to input if returned by error
1260 GOSUB 8270 'intro text
1270 GOSUB 2000 'name of substance
1280 GOSUB 8550 'load data from file, if any
1290 GOSUB 2030 'enter phys props
1300 GOSUB 2150 'calc for substance
1310 GOSUB 6170 'calculation options
1320 IF TN=0 THEN GOSUB 2360:GOSUB 4500 ELSE GOSUB 2500
'calc for temp of interest
1330 GOSUB 6400 'what next
1340 ON NC GOTO 1360,1310,1370
1350 RETURN
1360 CLS:GOSUB 5000:GOTO 1330
1370 FOR K=0 TO 2:FOR J=1 TO 8:A(K,J)=0:NEXT J,K:LP=0:GOTO 1270
1490 REM Mixture routine
1500 IF NE THEN 1540 'to entry if return on error
1510 GOSUB 8470 'intro blurb
1520 GOSUB 3000 'get names of materials
1530 IF NL=2 THEN GOSUB 3120 'range of calcs?
1540 GOSUB 3250 'data in
1550 GOSUB 3490 'estimate missing viscosities
1560 GOSUB 3750 'viscosity of mixture/print
1570 GOSUB 6510 'what next?
1580 ON NC GOTO 1610,1530,1650
1590 RETURN
1600 REM Hardcopy results
1610 IF AL THEN CLS:PRINT"Too late! Cannot print results for range now.":GOSUB
7070:GOTO 1570
1620 CLS:GOSUB 5410 'hardcopy results
1630 GOTO 1570
1640 REM Clean out data
1650 AL=0:FOR J=1 TO NL:FOR K=0 TO 2:A(K,J)=0:NEXT K,J:GOTO 1520
1990 REM Input material data
2000 CLS:KS=1
2010 INPUT"Name of Substance";YB
2020 RETURN
2030 CLS:NE=0 'reset error flag
2040 KS=1:KL=1:KM=8:ZA="PROPERTIES OF "+YB
2050 ZB="":ZD="":ZG="":J8=1
2060 IF NM=2 THEN KM=7 'no f. pt needed for St & Th
2070 FOR J=1 TO KM:YT(J)=Y(J):NEXT J 'instal titles
2080 NT=9-NM 'no of titles
2090 GOSUB 410 'to data input
2100 GOSUB 2600 'check input data valid
2110 IF NE THEN NE=0:GOTO 2090 ELSE GOSUB 8850
2120
MW=A(2,1):TC=A(2,2):PC=A(2,3):VC=A(2,4):W=A(2,5):TF=A(2,8)+C1:SG=A(2,6):TD=A(2
,7)+C1 'get data in equation variables
2130 RETURN
2140 REM Calculate basic parameters
2150 CLS
2160 TR=TD/TC:FZ=.29056-.08775*W 'Yamada & Gunn, 1973
2170 GOSUB 2290
2180 VS=MW/SG/F2 'calc scaling factor
2190 IF NM=2 THEN GOTO 2320 'Stiel and Thodos
2200 REM Pr and Sr method
2210 TR=TF/TC
2220 GOSUB 2290
2230 VM=F2*VS
2240 VZ=.0085*TC*W-2.02+VM/(.342*TR+.894)
2250 F1=4.27+.032*MW-.077*PC+.014*TF-3.82*TR
2260 B=.33*VC/F1 - 1.12
2270 RETURN
2280 REM Calc f2(t)
2290 F2=FZ^((1-TR)^C2) 'Yamada & Gunn 1973
2300 RETURN
2310 REM Stiel and Thodos parameter
2320 ES=TC^(1/6)
2330 ES=ES/SQR(MW)/(PC^(2/3))
2340 RETURN
2350 REM Calc for specific temp
2360 TR=(TL+C1)/TC
2370 GOSUB 2290:V=F2*VS
2380 IF NM=2 THEN GOTO 2420 'to Stiel and Thodos
2390 ET=B*(V/VZ-1):ET=1/ET
2400 GOTO 2480

```



```

2410 REM Stiel and Thodos calc
2420 IF TR<1.5 THEN UB=34E-05*(TR^.94) ELSE UB=17.78E-05*((4.58*TR-
1.67)^(5/8)) 'calc low press visc
2430 RD=VC/V
2440 UT=(((.0093324*RD-.040758)*RD+.058533)*RD+.023364)*RD+.1023
2450 UT=UT*UT*UT*UT
2460 UT=UT-1E-04+UB
2470 ET=UT/ES
2480 RETURN
2490 REM Calc for range of temps
2500 CLS
2510 FOR TL=TS TO TE STEP TN
2520 GOSUB 2360
2530 GOSUB 5170
2540 GOSUB 4650
2550 NEXT
2560 TL=TL-TN 'restore last temp calc'd
2570 IF N THEN GOSUB 7070
2580 RETURN
2590 REM Check validity of input data
2600 FOR J=KS TO KS+5
2610 IF A(2,J)>0 THEN 2630
2620 CLS:PRINT Y(J-KS+1);" cannot be zero or negative.":NE=1
2630 NEXT
2640 IF NE THEN PRINT:PRINT"Program will return to data entry. Please correct
input.":GOSUB 7070
2650 RETURN
2990 REM Initial input for mixtures
3000 CLS:JP=0
3010 NL=0 'initialise - NL no of comps
3020 NL=NL+1:YN(NL)="":MC(NL)=0:PRINT "Name of component no:":NL "- just
<ENTER> if no more components.":INPUT YN(NL)
3030 IF YN(NL)<>" THEN IF NL<10 GOTO 3020
3040 NL=NL-1:IF NL<2 THEN PRINT "Must have at least two components.":GOTO 3020
3050 CLS
3060 PRINT "Are your compositions in wt% or mol% - type <W> or <M>"
3070 YR="WwMm":GOSUB 7040
3080 MX=(KA>2) 'mx true if mol%
3090 IF MX THEN YU="mol%" ELSE YU="wt%"
3100 RETURN
3110 REM Calculate for range
3120 CLS:AL=0:MP=0:PRINT "Do you want to automatically calculate viscosities
for a range of compositions Y/N?"
3130 GOSUB 7030
3140 IF KA>2 THEN 3230
3150 PRINT "Minimum value of ";YU" for ";YN(1):INPUT AL
3160 IF (AL<0) OR (AL>100) THEN PRINT YU;"must be in range 0-100":GOTO 3150
3170 PRINT "Maximum value of ";YU" for ";YN(1):INPUT AH
3180 IF (AH<0) OR (AH>100) OR (AH<AL) THEN PRINT YU;"must be in range 0-100
and max must be bigger than min.":GOTO 3170
3190 PRINT "Increments of ";YU" for ";YN(1):INPUT AD
3200 PRINT:PRINT "Do you want hardcopy of the results as they are done, Y/N?"
3210 GOSUB 7030
3220 MP=(KA<3) 'true if hardcopy needed
3230 RETURN
3240 REM Initial data entry for mixtures
3250 FOR J=1 TO NL:IF LEN(YN(J))>NS-51 THEN YT(J+1)=LEFT$(YN(J),NS-51) ELSE
YT(J+1)=YN(J)
3260 NEXT
3270 YT(1)=STRING$(NS-46," ")+ "Visc - cp"+STRING$(7," ")+ "Mol.Wt"+STRING$(10,"
")+YU
3280 ZA="MIXTURE VISCOSITY INPUT DATA"
3290 ZB="LEAVE ANY UNKNOWN DATA AS 0 - MW NOT NEEDED IF USING MOL%"
3300 ZD="1,":ZG="":J8=3:NT=NL+1:KL=2:KM=NL+1:KS=0
3310 IF AL THEN A(2,1)=AL:A(2,2)=100-AL
3320 GOSUB 410 'screen routine
3330 GOSUB 3390 'check data
3340 IF NE THEN NE=0:GOTO 3320 'return if data check not OK
3350 FOR J=1 TO NL:AV(J)=A(0,J):AM(J)=A(1,J):AX(J)=A(2,J):NEXT 'put data into
working vars; av=visc, am=mw, ax=frn
3360 TL=-1000 'flag to show temp not set
3370 RETURN
3380 REM Check screen input for mixtures
3390 CLS:NE=0:A=0
3400 FOR J=1 TO NL

```

(continued)

```

3410 IF NOT MX THEN IF A(1,J)<=0 THEN PRINT"Molecular wt for ";YN(J);"
required":NE=1
3420 IF NL<>2 OR AL=0 THEN IF A(2,J)<0 OR A(2,J)>100 THEN PRINT YU;" of
";YN(J);" out of range":NE=1
3430 A=A+A(2,J)
3440 NEXT
3450 IF A>100 OR A<100 THEN PRINT"Components do not add to 100 ";YU:NE=1
3460 IF NE THEN GOSUB 7070
3470 RETURN
3480 REM Estimate viscosity of missing items
3490 CLS:NK=0 'nk is temperature flag
3500 FOR LM=1 TO NL 'lm is main counter
3510 IF AV(LM) THEN 3720 'skip if visc given
3520 MC(LM)=1:PRINT "Estimating viscosity of ";YN(LM)
3530 PRINT:PRINT "Is ";YN(LM):PRINT "a non-associating compound, Y/N?";
3540 GOSUB 7030
3550 IF KA<3 THEN NM=1 ELSE NM=2
3560 YB=YN(LM):KS=LM*8+4:A(2,KS)=AM(LM):GOSUB 8550 'check for data on disk
3570 CLS:NE=0 'reset error flag
3580 KL=1:KM=8:ZA="PROPERTIES OF "+YB
3590 ZB="":ZD="":ZG="":J8=1
3600 IF NM=2 THEN KM=7 'no f. pt needed for St & Th
3610 FOR J=1 TO KM:YT(J)=Y(J):NEXT 'instal titles
3620 NT=9-NM 'no of titles
3630 GOSUB 410 'to data input
3640 GOSUB 2600 'check input data valid
3650 IF NE THEN NE=0:GOTO 3630 ELSE GOSUB 8850
3660
MW=A(2,KS):TC=A(2,KS+1):PC=A(2,KS+2):VC=A(2,KS+3):W=A(2,KS+4):TF=A(2,KS+7)+C1:
SG=A(2,KS+5):TD=A(2,KS+6)+C1 'get data in equation variables
3670 GOSUB 2150 'start calculation
3680 CLS:IF NK=0 THEN NK=1:INPUT "Temperature of estimate, deg C.":TL
3690 GOSUB 2360
3700 AV(LM)=ET:A(0,LM)=CSNG(ET) 'save answer
3710 CLS:PRINT "Viscosity of ";YN(LM);" calculated":GOSUB 7010:NM=3
3720 NEXT
3730 RETURN
3740 REM Calculate visc of mixture
3750 IF AL THEN GOTO 3790
3760 IF NOT MX THEN GOSUB 3890 'conv to moles if nec
3770 GOSUB 3960 'calculate mixture visc
3780 GOTO 3860 'to return
3790 N=0:FOR A=AL TO AH STEP AD
3800 AX(1)=A:AX(2)=100-A
3810 IF NOT MX THEN GOSUB 3890
3820 GOSUB 3960
3830 IF MP THEN GOSUB 5570
3840 GOSUB 4730
3850 NEXT
3860 IF NL>2 OR AL=0 THEN GOSUB 4550 ELSE IF N THEN GOSUB 7070 'to screen if
results requ'd
3870 RETURN
3880 REM Convert to mole frn
3890 AT=0:FOR J=1 TO NL
3900 AY(J)=AX(J)/AM(J):AT=AT+AY(J)
3910 NEXT
3920 FOR J=1 TO NL
3930 AX(J)=AY(J)*100/AT
3940 NEXT:RETURN
3950 REM Visc of mixture
3960 AT=0
3970 FOR J=1 TO NL
3980 AT=AT+AX(J)*LOG(AV(J))
3990 NEXT
4000 ET=EXP(AT/100) 'viscosity of mixture
4010 RETURN
4490 REM Print results
4500 CLS
4510 PRINT "Viscosity of ";YB;" at";:PRINT USING " ###.# deg C is ###.###
cp";TL,ET
4520 PRINT
4530 RETURN
4540 REM Screen print for mixture
4550 CLS
4560 PRINT "Viscosity of mixture of:"
4570 FOR J=1 TO NL

```



```

4580 PRINT TAB(5) USING Y1;A(2,J);:PRINT " ";YU;" ";YN(J)
4590 NEXT
4600 PRINT:IF TL<>-1000 THEN PRINT "at ";TL;" deg C";
4610 PRINT " is";:PRINT USING Y3;ET;:PRINT " cp"
4620 GOSUB 7070
4630 RETURN
4640 REM Screen print list of results
4650 IF TL<>TS AND N<>0 THEN 4680
4660 N=0:PRINT "Viscosity of ";YB
4670 PRINT TAB(5) "Temperature, deg C";TAB(NS-25)"Viscosity, cp"
4680 PRINT TAB(12) USING Y2;TL;:PRINT TAB(NS-23) USING Y4;ET
4690 N=N+1
4700 IF N=NR-4 THEN GOSUB 7070:N=0:CLS
4710 RETURN
4720 REM Screen display list of mixture results
4730 IF A<>AL AND N<>0 THEN 4760 'jump titles if not first time
4740 CLS:N=0:PRINT TAB(12);YU;" of";TAB(54)"Viscosity"
4750 PRINT YN(1);TAB(25) YN(2);TAB(58)"cp"
4760 PRINT USING Y1;A;
4770 PRINT TAB(25) USING Y1;100-A;
4780 PRINT TAB(56) USING Y3;ET
4790 N=N+1
4800 IF N=NR-4 THEN GOSUB 7070:N=0
4810 RETURN
4990 REM Hardcopy results
5000 IF TN THEN CLS:PRINT "Too late! Cannot print results for range
now.":GOSUB 7070:GOTO 5150
5005 GOSUB 6000
5010 CLS:ON JP GOTO 5020,5090
5020 GOSUB 5290:GOSUB 5330
5030 LPRINT
5040 LPRINT TAB(5) "viscosity of ";YB;" estimated by method of ";YM(NM);" is:"
5050 LPRINT:LPRINT TAB(25) USING Y4+" cp at "+Y2+" deg C";ET,TL
5060 LPRINT:LPRINT
5070 GOTO 5150
5080 REM Table heading
5090 IF LT THEN 5140 'LT is flag for table
5100 GOSUB 5290
5110 LPRINT
TAB(5)"Substance";TAB(25)"Temperature";TAB(40)"Viscosity";TAB(52)"Method"
5120 LPRINT TAB(30)"deg C";TAB(44)"cp"
5130 LPRINT:LT=1
5140 LPRINT TAB(5)YB;:LPRINT TAB(29) USING Y2;TL;:LPRINT TAB(40) USING
Y4;ET;:LPRINT TAB(52) YM(NM)
5150 RETURN
5160 REM List results for temp range on printer
5170 IF LP=0 THEN 5270
5180 IF TL<>TS THEN 5250
5190 GOSUB 5290:GOSUB 5330
5200 LPRINT TAB(5)"Estimated viscosities using method of ";YM(NM)
5210 LPRINT
5220 LPRINT TAB(10)"Temperature";TAB(45)"Viscosity"
5230 LPRINT TAB(13)"deg C";TAB(49)"cp"
5240 LPRINT
5250 LPRINT TAB(14) USING Y2;TL;:LPRINT TAB(45) USING Y4;ET
5260 IF TL+TN>TE THEN LPRINT
5270 RETURN
5280 REM Print heading
5290 LPRINT TAB(5)"Project: ";YP;TAB(55)YD
5300 LPRINT
5310 RETURN
5320 REM Input data print
5330 LPRINT TAB(5)"INPUT DATA for ";YB
5340 LPRINT
5350 FOR J=1 TO 9-NM
5360 LPRINT TAB(10) Y(J);:LPRINT TAB(45) USING Y4;A(2,J)
5370 NEXT
5380 LPRINT
5390 RETURN
5400 REM Hardcopy for mixtures
5410 GOSUB 5290
5420 LPRINT TAB(40)"INPUT DATA":LPRINT
5430 LPRINT TAB(5)"Substance";TAB(35)"Visc - cp";TAB(48)"Mol. wt";TAB(63)YU
5440 FOR J=1 TO NL
5450 LPRINT TAB(5) YN(J);

```

(continued)

```

5460 LPRINT TAB(35) USING Y3;AV(J);
5470 IF MC(J) THEN LPRINT"*";
5480 LPRINT TAB(48) USING Y1;AM(J);
5490 LPRINT TAB(61) USING Y1;A(2,J)
5500 NEXT
5510 LPRINT:LPRINT TAB(10) "* means estimated value, not input"
5520 LPRINT:LPRINT TAB(5) "Viscosity of mixture";
5530 IF TL<>-1000 THEN LPRINT" at";TL;" deg C";
5540 LPRINT" is";:LPRINT USING Y3;ET;:LPRINT" cp.:LPRINT
5550 RETURN
5560 REM Table of mixture viscosities
5570 IF A<>AL THEN 5700 'only print title first time
5580 GOSUB 5290
5590 LPRINT:LPRINT TAB(5) "Viscosities of mixtures of:"
5600 LPRINT:FOR J=1 TO 2
5610 LPRINT TAB(10);YN(J)
5620 NEXT
5630 LPRINT
5640 IF TL<>-1000 THEN LPRINT TAB(5)" at";TL;" deg C ";
5650 LPRINT TAB(5) "are as follows:"
5660 LPRINT:LPRINT TAB(17);YU;" of ";
5670 LPRINT TAB(55)"Viscosity"
5680 LPRINT TAB(5);YN(1);TAB(30);YN(2);TAB(58)"cp"
5690 LPRINT
5700 LPRINT TAB(5) USING Y1;A;:LPRINT TAB(30) USING Y1;100-A;
5710 LPRINT TAB(55) USING Y3;ET
5720 IF A+AD>AH THEN LPRINT
5730 RETURN
5990 REM Hardcopy option menu
6000 CLS:IF JP=2 THEN 6060
6010 LOCATE 1,(NS-7)/2:PRINT "PRINT OPTIONS"
6020 PRINT:PRINT TAB(5)"1) Print just the current result complete with input
data"
6030 PRINT TAB(5)"2) Print table headings and start table for all":PRINT
TAB(8)"results until new project is started."
6040 NC=2:GOSUB 7120
6050 JP=NC 'JP = flag for choice
6060 RETURN
6070 REM Main menu
6080 CLS
6090 LOCATE 1,(NS-4)/2:PRINT "MAIN MENU"
6100 PRINT:PRINT TAB(5)"1) Viscosity of non-associating liquid - most
accurate"
6110 PRINT TAB(5)"2) Viscosity of associating liquid - not very accurate"
6120 PRINT TAB(5)"3) Viscosity of liquid mixture"
6130 NC=3:GOSUB 7120
6140 NM=NC
6150 RETURN
6160 REM Calculation options
6170 CLS
6180 LOCATE 1,(NS-10)/2:PRINT "CALCULATION OPTIONS"
6190 PRINT:PRINT TAB(5)"1) Estimate viscosity for one specific temperature"
6200 PRINT TAB(5)"2) Estimate viscosity for a range of temperatures"
6210 PRINT
6220 NC=2:GOSUB 7120
6230 TN=0
6240 ON NC GOTO 6370
6250 CLS:INPUT"Start of temperature range, deg C";TS
6260 INPUT"End of temperature range, deg C";TE
6270 INPUT"Temperature interval in range, deg C";TN
6280 PRINT:PRINT "Everything correct, Y/N? ";
6290 GOSUB 7030
6300 IF KA>2 THEN 6250
6310 CLS:PRINT "Do you want results tabulated on printer, Y/N? "
6320 GOSUB 7030
6330 LP=(KA<3)
6340 IF LP THEN PRINT "Press <ENTER> when printer is ready" ELSE 6380
6350 Y=INKEY$:IF Y="" THEN 6350
6360 GOTO 6380
6370 CLS:INPUT"Temperature, deg C";TL
6380 RETURN
6390 REM Menu after calc
6400 PRINT TAB((NS-4)/2),"WHAT NEXT"
6410 PRINT
6420 PRINT TAB(5)"1) Hardcopy of result (one result only). "
6430 PRINT TAB(5)"2) Viscosity for same material at another temperature."

```



```

6440 PRINT TAB(5)"3) Viscosity for different substance of the same type."
6450 PRINT TAB(5)"4) Main menu."
6460 PRINT TAB(5)"5) New project (all data lost)."
6470 PRINT TAB(5)"6) End program."
6480 NC=6:GOSUB 7120
6490 RETURN
6500 REM Menu after mix
6510 PRINT TAB((NS-4)/2),"WHAT NEXT"
6520 PRINT
6530 PRINT TAB(5)"1) Hardcopy of results"
6540 PRINT TAB(5)"2) Another calculation for same materials."
6550 PRINT TAB(5)"3) Calculate for a different mixture."
6560 PRINT TAB(5)"4) Main menu."
6570 PRINT TAB(5)"5) New project (all data lost)."
6580 PRINT TAB(5)"6) End program."
6590 NC=6:GOSUB 7120
6600 RETURN
6610 REM Project Intro routine
6620 CLS:INPUT"Enter project title";YP
6630 YD=LEFT$(DATE$,10):PRINT "Project date (default = ";YD;")";:INPUT YD
6640 RETURN
6990 REM Subroutines
7000 REM Delay
7010 FOR J=1 TO 2000:NEXT:RETURN
7020 REM Check inkey
7030 YR="YyNn"
7040 Y=INKEY$:IF Y="" THEN 7040
7050 KA=INSTR(YR,Y):IF KA THEN RETURN ELSE 7040
7060 REM Press key to cont
7070 LOCATE NR,1:PRINT "PRESS ANY KEY TO CONTINUE";
7080 Y=INKEY$:IF Y="" THEN 7080
7090 CLS
7100 RETURN
7110 REM Select from menu
7120 PRINT:PRINT"Choose by number"
7130 Y=INKEY$:IF Y="" THEN 7130
7140 Y%=VAL(Y)
7150 IF Y%<1 OR Y%>NC THEN 7130
7160 PRINT Y:NC=Y%
7170 RETURN
7180 REM Error trap
7190 CLS
7200 IF ERR=5 THEN 7250
7210 IF ERR=11 THEN 7290
7220 IF ERR=54 THEN 7360
7230 IF ERR=6 THEN 7270
7240 PRINT "Error in line";ERL:ON ERROR GOTO 0:RESUME
7250 PRINT "Illegal function call error has occurred in line";ERL:GOSUB 7340
7260 PRINT "Look for negative or zero values of input data":GOTO 7310
7270 PRINT "Overflow error has occurred in line";ERL:GOSUB 7340
7280 PRINT "Look for very low or very high values of input data":GOTO 7310
7290 PRINT "Division by zero error has occurred in line";ERL:GOSUB 7340
7300 PRINT "Look for very low or zero values of input data"
7310 PRINT "Check that input data units are correct, and that all values
are within the range of the correlation"
7320 GOSUB 7070:NE=1 'ne is error flag
7330 RESUME 1030
7340 PRINT "Program will return to data entry to allow you to check validity of
your input data"
7350 RETURN
7360 PRINT "No such file"
7370 GOSUB 7070:CLOSE:JF=1:RESUME NEXT
7990 REM Introductory blurb
8000 CLS
8010 GOSUB 8220:IF NB THEN 8200
8020 CLS:PRINT "This program estimates viscosities of pure liquids and
mixtures"
8030 PRINT:PRINT "The viscosities of pure, non-associating, liquids are
estimated by the";CHR$(13);"method of J.W.Przedzicki and T.Sridhar published
in the AIChEJ,"
8035 PRINT "Feb. 1985, p333. It is the most accurate method and should be
used";CHR$(13);"if possible."
8040 PRINT:PRINT "Data required for the estimate are the molecular weight, the
critical";CHR$(13);"properties, the acentric factor, and at least one value of
density at a";CHR$(13);"known temperature."

```

(continued)


```

8050 PRINT:PRINT "This method gives estimates with an average error of 9%, and
a maximum error of 40%, but results are very unreliable if used for
associating liquids."
8060 PRINT:GOSUB 7070
8070 CLS
8080 PRINT"For associating liquids there is really no reliable predictive
method."
8090 PRINT:PRINT "The method used in this program is that of Stiel and Thodos,
as described";CHR$(13);"in 'The Properties of Gases and Liquids' by Reid and
Sherwood, 2nd Ed., p437."
8100 PRINT:PRINT "Data required for the estimate are the molecular weight, the
critical";CHR$(13);"properties, the acentric factor, and at least one value of
density at a";CHR$(13);"known temperature."
8110 PRINT:PRINT "This method gives quite large errors (as much as -80%),
usually on the low";CHR$(13);"side, for some associating liquids, but no other
method is any more accurate."
8120 PRINT:GOSUB 7070
8130 CLS
8140 PRINT "Prediction of viscosities for liquid mixtures from pure component
data is";CHR$(13);"also rather unreliable."
8150 PRINT:PRINT "The method used in this program is procedure 8H from the
AIChE 'Data";CHR$(13);"Prediction Manual'. It is appropriate for systems
where the mixtures have";CHR$(13);"viscosities intermediate between those of
the components."
8160 PRINT:PRINT "Data required for the estimate are either the viscosities of
the components, or the data listed previously to allow them to be estimated.
Liquid";
8170 PRINT "compositions may be entered in mol% or wt%, but in the latter case
the";CHR$(13);"molecular weight is also required."
8180 PRINT:PRINT "This method is not suitable for systems that have maxima or
minima in the";CHR$(13);"mixture properties."
8190 GOSUB 7070
8200 RETURN
8210 REM Routine to bypass blurbs
8220 PRINT "This program includes explanatory text and
instructions.";CHR$(13);"If you are familiar with the program, you can bypass
these.";CHR$(13);"Do you want instructions displayed for this run, Y/N?"
8230 GOSUB 7030
8240 NB=(KA>2) 'nb is flag indicating No Blurbs
8250 RETURN
8260 REM Blurb for P&S method
8270 CLS:IF NB THEN 8450
8280 IF NM=2 THEN GOTO 8370
8290 PRINT "This calculation method gives quite accurate results for most"
8300 PRINT "non-associating compounds. Any errors tend to be on the low"
8310 PRINT "side, and the errors are more pronounced at lower temperatures,"
8320 PRINT "reaching a maximum close to the freezing point."
8330 PRINT:PRINT "The only common organics that cannot be handled are
alcohols,"
8340 PRINT "both aliphatic and aromatic. Cyclic and branched compounds
":PRINT"tend to give quite large errors, usually on the low side."
8350 PRINT "Organic acids seem to be predictable."
8360 GOTO 8440
8370 CLS
8380 PRINT "This calculation method gives reasonable results for many"
8390 PRINT "compounds. It works best at temperatures from"
8400 PRINT "the boiling point up, and the errors are more pronounced":PRINT
"at lower temperatures."
8410 PRINT "Only use this method for associating compounds."
8420 PRINT:PRINT "For all other substances, the Przezdziecki and Sridhar"
8430 PRINT "method is better."
8440 GOSUB 7070
8450 RETURN
8460 REM Blurb for mixtures
8470 IF NB THEN 8530
8480 CLS
8490 PRINT "This section calculates the viscosity of liquid mixtures from
the";CHR$(13);"viscosities of the pure components, for up to 10 components."
8500 PRINT:PRINT "If the pure component viscosities are not known, they will
be estimated by";CHR$(13);"one of the methods available. Leaving the
viscosity value as zero during"
8505 PRINT"data entry will automatically activate the estimating procedure."
8510 PRINT:PRINT "If there are only two components, a range of mixtures can be
calculated";CHR$(13);"automatically - you will be prompted for the range if
this option is chosen."
8520 GOSUB 7070

```



```

8530 RETURN
8540 REM Read data from file
8550 CLS
8560 PRINT"Checking disk for data..."
8570 GOSUB 8780:GOSUB 8670:GOSUB 8720
8580 IF N>LOF(1)/LF THEN 8650
8590 A(2,KS)=CVS(MF$):A(2,KS+1)=CVS(TF$)
8600 A(2,KS+2)=CVS(PF$):A(2,KS+3)=CVS(VF$)
8610 A(2,KS+4)=CVS(WF$):A(2,KS+5)=CVS(DF$)
8620 A(2,KS+6)=CVS(SF$):A(2,KS+7)=CVS(FF$)
8630 FOR J=1 TO 7:IF ABS(A(2,KS+J))<C3 THEN A(2,KS+J)=0
8640 NEXT
8650 RETURN
8660 REM Open file for crit data
8670 Y=LEFT$(YH,1)+"CRIT.DAT"
8680 OPEN"R",1,Y,128
8690 FIELD 1,32 AS NF$,4 AS MF$,4 AS BF$,4 AS TF$,4 AS PF$,4 AS VF$,4 AS WF$,4
AS DF$,4 AS SF$,4 AS FF$,60 AS XF$
8700 RETURN
8710 REM Find if name in record
8720 N=1
8730 GET 1,N
8740 J=LEN(YH):K=LEN(NF$):IF LEFT$(NF$,J)=YH THEN IF RIGHT$(NF$,K-
J)=STRING$(K-J," ") THEN 8760
8750 N=N+1:IF N<=LOF(1)/LF THEN 8730
8760 RETURN
8770 REM Convert name to caps
8780 YH=YB
8790 FOR J=1 TO LEN(YB)
8800 K=ASC(MID$(YB,J))
8810 IF K>96 AND K<123 THEN YL=CHR$(K-32):MID$(YH,J)=YL
8820 NEXT
8830 RETURN
8840 REM Write data to disk
8850 IF N<=LOF(1)/LF THEN LSET FF$=MKS$(A(2,KS+7)):GOTO 8920
8860 LSET NF$=YH
8870 LSET MF$=MKS$(A(2,KS)):LSET TF$=MKS$(A(2,KS+1))
8880 LSET PF$=MKS$(A(2,KS+2)):LSET VF$=MKS$(A(2,KS+3))
8890 LSET WF$=MKS$(A(2,KS+4)):LSET DF$=MKS$(A(2,KS+5))
8900 LSET SF$=MKS$(A(2,KS+6)):LSET FF$=MKS$(A(2,KS+7))
8910 LSET BF$=" "
8920 PUT 1,N
8930 CLOSE 1
8940 RETURN

```

critibm.bas

"Small-Scale Engineering Applications," by J. Neil
Stone. July, page 253.

```

0 REM Predict critical properties - Apr 85 - CRITIBM.BAS. Version for BASICA
10 REM Rummens & Rajan, Can.J.Ch.E., Jun 1979, 349
20 REM Data input screen concept by L.E.Sparks, ACCESS, 1982 p 10.
30 REM Edited for BYTE Dec 1985
40 GOTO 170
50 DATA"***** CRITICAL PROPERTY *****"
60 DATA"*** PREDICTION ***"
70 DATA" by"
80 DATA" J.Neil Stone"
90 DATA" Ledge Engineering Inc"
100 DATA" 179 Lansdowne Avenue"
110 DATA" Kingsville, Ontario, N9Y 3J2"
120 DATA" "
130 DATA" **** Apr 1985 ****"
140 CLS:FOR J=1 TO 9:READ Y:LOCATE NR/2-5+J,NS/2-17:PRINT Y:NEXT
150 GOSUB 6250:RETURN
160 REM Initialize
170 CLEAR 500
180 DEFINT J-L,N:DEFSTR R,X-Z
190 DIM Y,K,J,M,TD,Y%,LF,KA
200 'POKE 16409,1 'Set caps mode
210 'NS=64:NR=16:LF=1 'parameters for TRS80
220 NS=80:NR=24:LF=128 'parameters for 80x24

```

(continued)

```

230 DEF FN LC(J,K)=(J-1)*NS+K-1
240 GOSUB 50 'display title
250 DIM PB(3),E(2,11),Y(12),P(2),TR(2),F(1),V(2) 'arrays
260 REM Set up message strings
270 Y(1)="Molecular weight":Y(5)="B.Pt,":Y(2)=Y(5)+" deg C":Y(3)="Specific
gravity, 20/4":Y(4)=" (opt)":Y(6)="Vap.Press, ":ZZ(8)="Density,
g/cc":Y(8)="deg F":Y(9)="deg C":Y(10)="mmHg":Y(11)="psia":Y(12)="kPa"
280 Y1="###.##":Y2="###.###":Y3="###.###"
290 Z="-->":Z1=CHR$(10)+CHR$(91)+CHR$(9)+CHR$(13)+"X"+"X":Z3=" ":Z4=" "+
0123456789":Z2=Z4+Z1:Z5=CHR$(8)+CHR$(13)+"ED"+Z4
300 Z8="USE UP, DN, RT ARROWS TO SELECT ENTRY. <X> TO EXIT":Z9="USE <-- TO
CORRECT. PRESS <ENTER> WHEN DONE":ZA=STRING$(17," ")
310 REM Set up constants
320 C1=1.5:C2=36:C3=42:C4=35:C5=.3143:C6=.0838:C7=8.315E
03:C8=1.9:C9=.26:CA=2.38:CB=.2:CC=.489:CD=.225:CE=.511:CF=660:CG=.1:CH=1.8:CJ
=32:CK=.10135:CL=760:CM=14.696:CN=1000:CP=273.16
330 ON ERROR GOTO 6130
340 GOTO 1500 'to main program
490 REM Screen routine - put titles in ZZ()
500 CLS:LOCATE 1,(NS-LEN(RA))/2):PRINT RA
510 PRINT STRING$(NS-1," ")
520 FOR J=KL TO KM
530 IF J=7 THEN PRINT:GOTO 570
540 IF J=KL THEN PRINT TAB(5) ZZ(J) TAB(NS-14) E(2,J):GOTO 570
550 PRINT TAB(5) ZZ(J);
560 IF J>3 THEN PRINT TAB(NS-30) E(1,J) TAB(NS-14) E(2,J) ELSE PRINT
570 NEXT
580 LOCATE NR-3,1:PRINT STRING$(NS-1,95)
590 LOCATE NR-2,1:PRINT STRING$(NS-1," ");
595 LOCATE NR-2,1:PRINT RB;
600 KQ=NS-17:KP=KL+1:J9=3:GOTO 660
610 LOCATE KP+1,KQ+1:PRINT Z;
620 LOCATE NR-1,1:PRINT STRING$(NS-1," ");
625 LOCATE NR-1,1:PRINT Z8;
630 Y$=INKEY$:IF Y$="" THEN 630
640 IF LEN(Y$)=1 THEN IF INSTR(Z2,Y$)=0 THEN 630
650 IF LEN(Y$)=1 THEN J9=INSTR(Z1,Y$) ELSE J9=ASC(RIGHT$(Y,1)):IF J9=80 THEN
J9=1 ELSE IF J9=72 THEN J9=2 ELSE IF J9=77 THEN J9=3 ELSE 630
660 IF J9=0 THEN 800 ELSE LOCATE KP+1,KQ+1:PRINT Z3;:ON J9 GOTO
690,680,710,680
670 RETURN
680 J9=J9-3
690 KP=KP+J9:IF KP>KM+1 THEN KP=KL+1 ELSE IF KP<KL+1 THEN KP=KM+1
700 GOTO 750
710 ON J8 GOTO 750,740
720 IF KQ=NS-49 OR KQ=NS-33 THEN KQ=KQ+16 ELSE KQ=NS-49
730 GOTO 750
740 IF KQ=NS-33 THEN KQ=NS-17 ELSE KQ=NS-33
750 ZF=STR$(KP-1):ZF=RIGHT$(ZF,LEN(ZF)-1)+" ";
760 IF INSTR(RD,ZF) THEN IF J9=3 THEN J9=1:GOTO 690 ELSE 690
770 IF INSTR(RE,ZF) THEN KQ=NS-17
780 GOTO 610
790 REM Data entry section
800 LOCATE NR-1,1:PRINT STRING$(NS-1," ");
805 LOCATE NR-1,1:PRINT Z9;
810 KC=1:ZE=Y$
820 LOCATE KP+1,KQ+1:PRINT ZA;:LOCATE KP+1,KQ+1+4:PRINT ZE;CHR$(95);
830 Y$=INKEY$:IF Y$="" THEN 830
840 IF INSTR(Z5,Y$)=0 THEN IF LEN(Y$)>1 AND ASC(RIGHT$(Y$,1))=75 THEN 900 ELSE
830
850 ON INSTR(Z5,Y$) GOTO 900,920
860 KC=KC+1
870 ZE=ZE+Y$
880 LOCATE KP+1,KQ+1+KC+3:PRINT Y$;CHR$(95);
890 GOTO 830
900 IF KC=0 THEN 830 ELSE KC=KC-1:IF KC=0 THEN ZE="" ELSE ZE=LEFT$(ZE,KC)
910 GOTO 820
920 KE=(KQ-NS+49)/16
930 E(KE,KP-1)=VAL(ZE)
940 LOCATE KP+1,KQ+1:PRINT ZA;:LOCATE KP+1,KQ+1+3:PRINT CSNG(VAL(ZE));
950 ZE="":J9=1
960 IF KQ=NS-17 THEN KP=KP+1
970 IF KP>KM+1 THEN KP=KL+1
980 GOTO 710
1490 REM Main program

```



```

1500 GOSUB 6500      'introductory text
1510 GOSUB 4610      'get title etc
1520 GOSUB 3000      'get initial info
1530 GOSUB 2000      'input data
1540 GOSUB 2500      'calculate critprops
1550 GOSUB 3500      'results on screen
1560 GOSUB 4500      'What next?
1570 ON NC GOTO 1590,1600,1610,1610
1580 CLS:ON ERROR GOTO 0:END
1590 GOSUB 4000:GOTO 1560 ' to hardcopy routine
1600 GOSUB 5000:GOTO 1560 'save to disk
1610 FOR J=1 TO 2:FOR K=1 TO 8:E(J,K)=0:NEXT K,J 'clean out input array
1620 ON NC-2 GOTO 1510,1520
1990 REM Data input
2000 CLS
2010 RA="INPUT DATA FOR "+YA
2020 RB="DATA FOR SECOND VAPOUR PRESSURE POINT OPTIONAL"
2030 RD="2,3,7,"
2040 RE="1,"
2050 J8=2
2060 KL=1:KM=8:ON JF GOTO 2080,2080,2090
2070 E(1,4)=CK*CN:GOTO 2100
2080 E(1,4)=CL:GOTO 2100
2090 E(1,4)=CM
2100 ZZ(1)=Y(1):ZZ(2)=STRING$(NS-36," ")+ "Value"+STRING$(9,"
")+"Temperature":ZZ(3)=STRING$(43," ")+YT:ZZ(4)=Y(5)+
"+YP+"/"+"YT:ZZ(5)=Y(6)+" "+YP:ZZ(6)=ZZ(5):ZZ(7)=" "
2110 GOSUB 500 'to screen entry
2120 M=E(2,1)
2130 N=4
2140 FOR J=0 TO 2
2150 ON JF GOTO 2170,2180,2190
2160 P(J)=E(1,J+N)/CN:T(J)=E(2,J+N)+CP:C=CN*CK:GOTO 2200
2170 P(J)=E(1,J+N)*CK/CL:T(J)=E(2,J+N)+CP:C=CL:GOTO 2200
2180 P(J)=E(1,J+N)*CK/CL:T(J)=(E(2,J+N)-CJ)/CH+CP:C=CL:GOTO 2200
2190 P(J)=E(1,J+N)*CK/CM:T(J)=(E(2,J+N)-CJ)/CH+CP:C=CM
2200 NEXT
2210 SG=E(1,8):D=SG*CN
2220 IF JF=2 OR JF=3 THEN T=(E(2,8)-CJ)/CH ELSE T=E(2,8)
2230 T(3)=T+CP
2240 RETURN
2490 REM Calculation
2500 CLS
2510 PRINT"Calculating"
2520 IF P(2)=0 THEN N=0 ELSE N=1
2530 IF P(0)=0 THEN GOSUB 2710 ELSE TB=T(0):TD=T(0)
2540 T1=C1*TD
2550 FOR J=0 TO 1
2560 TR(J)=T(N+J)/T1
2570 F(J)=C2/TR(J)+C3*LOG(TR(J))-C4-TR(J)^6
2580 NEXT
2590 DF=F(0)-F(1)
2600 AL=(LOG(P(N)/P(N+1))-C5*DF)/(LOG(T(N)/T(N+1))-C6*DF)
2610 PC=EXP(LOG(P(N))-C5*F(0)-AL*(LOG(TR(0))-C6*F(0)))
2620 VC=C7*T1/PC/(C8+C9*AL)
2630 V(1)=VC/(CA+CB*AL)
2640 TR(2)=T(3)/T1:V(2)=M*(CC-CD*TR(2)+CE*((1-TR(2))^(1/3)))/D
2650 T2=T1+CF*(V(1)-V(2))
2660 IF ABS(T2-T1)<=CG THEN 2680
2670 T1=T2:GOTO 2550 'loop if not converged
2680 PB(0)=T1:PB(1)=PC:PB(2)=VC*CN:PB(3)=.2033*AL-1.1816
2690 RETURN
2700 REM Estimate bpt by Antoine
2710 B=LOG(P(1)/P(2))/(1/T(2)-1/T(1))
2720 A=LOG(P(1))+B/T(1)
2730 TD=B/(A-LOG(C))
2740 RETURN
2980 REM
2990 REM Compound name
3000 INPUT"Name of compound";YA
3010 IF JF THEN 3130 'jf is unit flag
3020 PRINT:PRINT"Which units are you using?"
3030 PRINT:PRINT"1) mmHg and deg C"
3040 PRINT"2) mmHg and deg F"
3050 PRINT"3) psia and deg F"

```

(continued)

```

3060 PRINT"4) kPa and deg C"
3070 NC=4:GOSUB 6070
3080 JF=NC:ON JF GOTO 3100,3110,3120
3090 YP=Y(12):YT=Y(9):GOTO 3130
3100 YP=Y(10):YT=Y(9):GOTO 3130
3110 YP=Y(10):YT=Y(8):GOTO 3130
3120 YP=Y(11):YT=Y(8)
3130 RETURN
3490 REM Display results on screen
3500 CLS
3510 YC="CRITICAL PROPERTIES OF "+YA:J=(NS-LEN(YC))/2
3520 PRINT TAB(J);YC
3530 PRINT
3540 J=(NS-40)/2:K=(NS+30)/2
3550 PRINT TAB(J)"Critical temperature, K:":PRINT TAB(K) USING Y1;PB(0)
3560 PRINT TAB(J)"Critical pressure, MPa:":PRINT TAB(K) USING Y3;PB(1)
3570 PRINT TAB(J+19)"atma:":PRINT TAB(K) USING Y1;PB(1)/CK
3580 PRINT TAB(J)"Critical volume, cc/mol:":PRINT TAB(K) USING Y1;PB(2)
3590 PRINT
3600 PRINT TAB(J)"Pitzer acentric factor :":PRINT TAB(K) USING Y2;PB(3)
3610 PRINT TAB(J)"Riedel critical parameter :":PRINT TAB(K) USING Y2;AL
3620 GOSUB 6030
3630 RETURN
3990 REM printer output
4000 CLS
4010 J=PEEK(14312) AND 240 'check if printer ready
4020 IF J=48 THEN 4060
4030 PRINT"Printer not on line - press <ENTER> when ready"
4040 Y=INKEY$:IF Y="" THEN 4040
4050 GOTO 4000
4060 IF LP THEN 4370
4070 PRINT"If you are going to calculate properties for several
substances":PRINT"the results can be printed as a table."
4080 PRINT"Otherwise they will be printed with each set as a
separate":PRINT"report."
4090 PRINT"Press <T> for tabulated results, <S> for separate reports";
4100 YR="TtTs":GOSUB 6010
4110 LP=(KA<3)
4120 LPRINT TAB(10)"Project: ";YB;TAB(70);YD:LPRINT
4130 IF LP THEN 4270
4140 REM Single printout
4150 J=(80-LEN(YC))/2:LPRINT TAB(J);YC
4160 LPRINT
4170 LPRINT TAB(18)"Critical temperature, K:":LPRINT TAB(55) USING Y1;PB(0)
4180 LPRINT TAB(18)"Critical pressure, MPa:":LPRINT TAB(55) USING Y3;PB(1)
4190 LPRINT TAB(37)"atma:":LPRINT TAB(55) USING Y1;PB(1)/.10133
4200 LPRINT TAB(18)"Critical volume, cc/mol:":LPRINT TAB(55) USING Y1;PB(2)
4210 LPRINT
4220 LPRINT TAB(18)"Pitzer acentric factor :":LPRINT TAB(55) USING Y2;PB(3)
4230 LPRINT TAB(18)"Riedel critical parameter :":LPRINT TAB(55) USING
Y2;AL:GOTO 4240
4240 LPRINT:LPRINT TAB(10)"These properties calculated using the method of
Rummen and Rajan":LPRINT TAB(10)"with the following input data:":LPRINT
4250 LPRINT TAB(18)"Molecular weight":LPRINT TAB(55) USING Y3;M
4260 IF P(0) THEN LPRINT TAB(18) Y(5)+" "+YT:LPRINT TAB(55) USING Y1;E(2,4)
4270 LPRINT TAB(18) ZZ(5):LPRINT TAB(45) USING Y2+" "+YP+" @"+Y1+"
"+YT;E(1,5),E(2,5)
4280 IF E(1,6) THEN LPRINT TAB(18) ZZ(6):LPRINT TAB(45) USING Y2+" "+YP+"
@"+Y1+" "+YT;E(1,6),E(2,6)
4290 LPRINT TAB(18)ZZ(8):LPRINT TAB(45) USING Y2+" g/cc"+" @"+Y1+"
"+YT;E(1,8),E(2,8)
4300 LPRINT CHR$(12)
4310 GOTO 4380
4320 REM Tabular printout
4330 LPRINT TAB(10)"Name";TAB(50)"Critical";TAB(70)"Acentric"
4340 LPRINT TAB(40)"Temp";TAB(50)"Pressure";TAB(62)"Volume";TAB(70)"factor"
4350 LPRINT TAB(40)" K";TAB(49)"MPa atm";TAB(62)"cc/mol"
4360 LPRINT
4370 LPRINT TAB(10);YA:LPRINT TAB(40) USING Y1;PB(0):LPRINT TAB(47) USING
Y3;PB(1):LPRINT TAB(55) USING Y1;PB(1)/.10133:LPRINT TAB(62) USING
Y1;PB(2):LPRINT TAB(70) USING Y2;PB(3)
4380 RETURN
4490 REM Menu
4500 CLS
4510 LOCATE 1,NS/2-3:PRINT "WHAT NEXT"

```



```

4520 PRINT:PRINT TAB(10)"1) Hardcopy of results"
4530 PRINT TAB(10)"2) Save critical properties to disk"
4540 PRINT TAB(10)"3) New project"
4550 PRINT TAB(10)"4) Calculate another material"
4560 PRINT TAB(10)"5) End program"
4570 NC=5:GOSUB 6070
4580 RETURN
4600 REM Project intro routine
4610 CLS:LP=0:JF=0:INPUT"Enter project title";YB
4620 YD=LEFT$(DATE$,10):PRINT"Project date (default = ";YD;")";:INPUT YD
4625 IF YD="" THEN YD=LEFT$(DATE$,10)
4630 RETURN
4990 REM Save critical data
5000 GOSUB 5240:GOSUB 5120:GOSUB 5170
5010 CLS:PRINT"Saving to file ";Y
5020 IF N>LOF(1)/LF THEN LSET XF$=STRING$(64," ")
5030 LSET NF$=YH
5040 LSET MF$=MKS$(M):LSET BF$=MKS$(TB)
5050 LSET TF$=MKS$(PB(0)):LSET PF$=MKS$(PB(1)/.10133)
5060 LSET VF$=MKS$(PB(2)):LSET WF$=MKS$(PB(3))
5070 LSET DF$=MKS$(SG):LSET SF$=MKS$(T)
5080 PUT 1,N
5090 CLOSE 1
5100 GOSUB 6250:RETURN
5110 REM Open file for crit data
5120 Y=LEFT$(YH,1)+"CRIT.DAT"
5130 OPEN"R",1,Y,128
5140 FIELD 1,32 AS NF$,4 AS MF$,4 AS BF$,4 AS TF$,4 AS PF$,4 AS VF$,4 AS WF$,4
AS DF$,4 AS SF$,64 AS XF$
5150 RETURN
5160 REM Find if name in record
5170 N=1
5180 IF N>LOF(1)/LF THEN 5220
5190 GET 1,N
5200 J=LEN(YH):K=LEN(NF$):IF LEFT$(NF$,J)=YH THEN IF RIGHT$(NF$,K
-J)=STRING$(K-J," ") THEN 5220
5210 N=N+1:GOTO 5180
5220 RETURN
5230 REM Convert name to caps
5240 YH=YA
5250 FOR J=1 TO LEN(YA)
5260 K=ASC(MID$(YA,J)) 5270 IF K>96 AND K<123 THEN YL=CHR$(K-32):MID$(YH,J)=YL
5280 NEXT
5290 RETURN
5990 REM Subroutines
6000 YR="YyNn"
6010 Y=INKEY$:IF Y="" THEN 6010
6020 KA=INSTR(YR,Y):IF KA THEN PRINT " ";Y:RETURN ELSE 6010
6030 LOCATE NR-1,1:PRINT "PRESS ANY KEY TO CONTINUE";
6040 Y=INKEY$:IF Y="" THEN 6040
6050 CLS
6060 RETURN
6070 PRINT:PRINT"Choose by number ";
6080 Y=INKEY$:IF Y="" THEN 6080
6090 Y%=VAL(Y)
6100 IF Y%<1 OR Y%>NC THEN 6080
6110 PRINT Y:NC=Y%
6120 RETURN
6130 CLS
6140 IF ERR=5 THEN 6160
6150 PRINT"Error in line";ERL:ON ERROR GOTO 0:RESUME
6160 PRINT"Math error has occurred in line";ERL
6170 PRINT"Program will return to data entry to allow you to re-enter your
input data."
6180 PRINT"Look for very low or very high, zero or negative values
of":PRINT"input data."
6190 PRINT"Check that input data units are correct, and that all values
are within the range of the correlation."
6200 GOSUB 6030: RESUME 1530
6240 REM Delay
6250 FOR J=1 TO 4000:NEXT:RETURN
6490 REM Opening text
6500 CLS
6510 PRINT"This program calculates the critical properties and
acentric":PRINT"factors for many compounds.":PRINT

```

(continued)

```

6600 PRINT "It is based on the correlation by Rummens and Rajan,
published":PRINT "in the Can.J.Ch.E., Jun 1979, (Vol 57, No.3), page 349."
6610 PRINT:PRINT "Input data required are the molecular weight, the vapor"
6620 PRINT "pressure at any two temperatures, and the density at
any":PRINT "temperature."
6630 PRINT:PRINT "One of the vapor pressure points may be the normal
boiling":PRINT "point, although the authors recommend against this,
and":PRINT "suggest two other points 40 to 60 K apart will give best results"
6640 GOSUB 6030
6650 RETURN

```

ascstr.asm

"Structural Analysis," by Robert W. Johnson and Fernando
G. Loygorri. July, page 199.

TITLE ASCSTR - FUNCTION TO DETERMINE ASCII CODE
PAGE ,132

; (C) Copyright Microstress Corporation 1984, 1985, 1986

COMMENT *

ASCSTR is a routine designed to be called from FORTRAN as a function
to return the ASCII code of the character specified by the input.

Mode of use:

code = ASCSTR (i,string)

where

code = value returned by the function, the requested ASCII code.
i = index to specify the location in the string of the character
whose ASCII code is requested.
string = name of the string (variable of type CHARACTER) where
the requested character is located.

*

SUBTTL FORMAL DECLARATIONS
PAGE

```

csascs SEGMENT 'CODE'
        ASSUME CS:csascs

```

SUBTTL ASCSTR - EXECUTABLE CODE
PAGE

```

PUBLIC ascstr
ascstr PROC FAR

```

```

        PUSH    BP
        MOV     BP,SP
        PUSH    ds
        LDS     BX,DWORD PTR [BP+10]
        MOV     CX,[BX]
; Check positive request.
        cmp     cx,0
        jle     outbounds
; Locate the end of the string (logical or physical)
        LDS     BX,DWORD PTR [BP+6]
scanend:
        mov     al,[bx]
        cmp     al,0
        je      outbounds
        cmp     al,6
        je      outbounds
        inc     bx
        loop    scanend
; Value is returned in AX.
        XOR     AH,AH
        jmp     exit
; Handle a request out of bounds.
outbounds:
        xor     ax,ax
exit:
        POP     ds

```



```

MOV    SP,BP
POP    BP
RET     8h

```

```

ascstr  ENDP
csascs  ENDS

```

```
END
```

```
test.seq
```

"Analog Circuit Analysis," by David McNeill.
July, page 170. See ac.prg for details. Commodore 64

```

%14
* TEST CIRCUIT
*
* FILE NAME IS TEST
*
R1 1 2 10K
R2 2 3 40K
R3 4 3 4K
C1 2 3 100PF
VIN 1 0 -2 DC 1 0 AC
VCC 4 0 20 DC
Q1 3 2 0 1
.MODEL 1 NPN BF=50 BR=1 IS=1E-14 VA=100 CJE=2PF CJC=4PF
.OUT 4 V(3) V(3)/(VIN) ZIN(VIN) ZOUT(3)
.END

```

```
testdc.seq
```

"Analog Circuit Analysis," by David McNeill.
July, page 170. See ac.prg for details.

7 ,	0 ,	0 ,	V(3)/(VIN),
6 ,	0 ,	0 ,	ZIN(VIN),
R1,	0 ,	0 ,	ZOUT(3),
R2,	3 ,	.02 ,	1 ,
R3,	-2 ,	1 ,	2 ,
C1,	1 ,	.01 ,	3 ,
VIN,	0 ,	1E-14 ,	4 ,
VCC,	5 ,	1 ,	0 ,
Q1,	1 ,	1 ,	0 ,
1 ,	0 ,	2E-12 ,	
10000 ,	3 ,	4E-12 ,	
1 ,	20 ,	6E-10 ,	
2 ,	4 ,	6E-09 ,	
0 ,	0 ,	1 ,	
0 ,	6 ,	1 ,	
0 ,	0 ,	1 , 0 , 0 ,	
1 ,	0 ,	4 ,	
40000 ,	5 ,	-5 ,	
2 ,	3 ,	3 ,	
3 ,	2 ,	0 ,	
0 ,	0 ,	0 ,	
0 ,	1 ,	5 ,	
0 ,	0 ,	0 ,	
1 ,	1 ,	3 ,	
4000 ,	1 ,	0 ,	
4 ,	0 ,	-2 ,	
3 ,	0 ,	5 ,	
0 ,	0 ,	0 ,	
0 ,	0 ,	0 ,	
0 ,	0 ,	-1 ,	
10 ,	0 ,	3 ,	
1E-10 ,	0 ,	0 ,	
2 ,	0 ,	0 ,	
3 ,	0 ,	V(3),	

truss2.bas

"Engineering on a Micro," Chris Pedicini. July, page 145.

```

10 '
20 ' PROGRAM = TRUSS2: VERSION 1.0: COPYRIGHT 1984 - C.S. PEDICINI
30 '
40 ON ERROR GOTO 22000
50 DEFINT I-N
60 GOSUB 20000
70 '
80 ' DIMENSION VARIABLES
90 '
100 DIM XNOD(100),YNOD(100),NCON(100,2),AR(100)
110 DIM A(200,25),B(200),STR(100)
120 DIM XYYY(4),BK(4,4)
130 DIM EKEY(5),NUM(10,10),NDIG(6),MARKER(10),ID(3,7),NMK(6)
140 DIM ILOAD(100,2),RLOAD(100),MENU$(4,7),PRNT$(5)
150 '
160 ' DATA FOR GRAPHICS
170 '
180 DATA 6,6,12288,18504,12360
190 DATA 6,6,8192,8224,8224
200 DATA 6,6,12288,4168,30752
210 DATA 6,6,30720,14344,30728
220 DATA 6,6,18432,30792,2056
230 DATA 6,6,30720,30784,30728
240 DATA 6,6,30720,30784,30792
250 DATA 6,6,30720,2120,2056
260 DATA 6,6,12288,12360,12360
270 DATA 6,6,30720,30792,30728
280 '
290 DATA 9,9,0,48,48,252,48,48,0,0
295 DATA 7,7,14336,-258,14590,0,0
300 '
310 ' DATA FOR SPREADSHEET EDITOR
320 '
330 DATA 1,8,10,10,10,20
340 DATA 1,8,6,6,6,18
350 DATA 1,8,1,1,10,18
360 '
370 DATA "MAIN MENU","DATA EDITOR","GRAPHICS","PRINT DATA",
"ANALYSIS/RESULTS", "GET/SAVE/DELETE","QUIT"
380 DATA "FEM DATA EDITOR","EDIT NODES","EDIT ELEMENTS","EDIT LOADS",
"FIND BANDWIDTH","CONTROL CARDS",""
390 DATA "GRAPHICS","ORIGINAL GEOMETRY","FINAL GEOMETRY","STRESS PLOT",
"","",""
400 DATA "GET/SAVE/DELETE","GET DATA.. FEM MODEL","SAVE DATA. FEM MODEL",
"DELETE","","",""
420 '
430 ' READ DATA
440 '
450 FOR I=1 TO 10: FOR J=0 TO 4: READ NUM(I,J): NEXT J: NEXT I
460 FOR I=0 TO 9: READ MARKER(I): NEXT I
465 FOR I=0 TO 6: READ NMK(I): NEXT I
470 FOR K=1 TO 3: FOR J=1 TO 6: READ ID(K,J) : NEXT J: NEXT K
480 FOR I=1 TO 4: FOR J=1 TO 7: READ MENU$(I,J): NEXT J: NEXT I
485 FOR I=1 TO 5: PRNT$(I)="Y": NEXT I
490 '
500 ' INITIALIZE VARIABLES
510 '
520 BLANK$=""
530 FKEY$=" ESC = RETURN TO MENU, F1 = DELETE LINE,
"+CHR$(27)+CHR$(24)+CHR$(25) +CHR$(26)+" POSITION CURSOR"
540 MAXNP=100: MAXEL=100: MAXLD=100: MBW=30
550 DV$="A:": DAT1$=DATE$: EXAG=1
560 EKEY(1)=30000000#: EKEY(2)=30000
570 IERR=0: NLOAD=0: NUMNP=0: NUMEL=0
572 FOR I=1 TO MAXEL
574 AR(I)=1!
576 NEXT I
580 '
590 ' MAIN MENU
600 '

```



```

610 IMENU=1: GOSUB 25000: IOPT=ISEL
620 IF IRET<>0 THEN GOTO 610
630 ON IOPT GOSUB 650,3460,31000,10000,5610,24010
640 GOTO 610
650 '
660 ' DATA EDITOR MENU
670 '
680 IMENU=2: GOSUB 25000: IOPT2=ISEL
690 IF IRET<>0 THEN RETURN
700 ON IOPT2 GOSUB 730,780,830,880,3070
710 GOTO 680
720 '
730 ' EDIT NODES
740 '
750 NCUR=NUMNP: NLMT=MAXNP: K=1: GOSUB 950
760 NUMNP=NCUR: ID(K,1)=NFRST: RETURN
770 '
780 ' EDIT ELEMENTS
790 '
800 NCUR=NUMEL: NLMT=MAXEL: K=2: GOSUB 950
810 NUMEL=NCUR: ID(K,1)=NFRST: RETURN
820 '
830 ' EDIT LOADS
840 '
850 NCUR=NLOAD: NLMT=MAXLD: K=3: GOSUB 950
860 NLOAD=NCUR: ID(K,1)=NFRST: RETURN
870 '
880 ' CALCULATE BANDWIDTH
890 '
900 CLS
910 GOSUB 30000
920 LOCATE 12,30: PRINT USING "MAX BANDWIDTH = ###";BW
930 LOCATE 16,27: PRINT "PRESS ANY KEY TO CONTINUE"
940 A1$=INKEY$: IF A1$="" THEN 940 ELSE RETURN
950 '
960 ' ** FULL SCREEN EDITOR **
970 '
980 NFRST=ID(K,1) ' NUMBER OF FIRST ROW IN ARRAY
990 NROW =ID(K,2) ' ROW POSITION OF CURSOR
1000 NCOL =ID(K,3) ' COLUMN POSITION OF CURSOR
1010 MNCOL=ID(K,4) ' MINIMUM COLUMN POSITION OF CURSOR
1020 NLEN =ID(K,5) ' LENGTH OF CURSOR
1030 MXCOL=ID(K,6) ' MAXIMUM COLUMN POSITION OF CURSOR
1040 GOSUB 1580: GOSUB 2140
1050 COLOR 7,0: LOCATE 24,1 : PRINT "      COMMAND =
";: LOCATE 24,15
1060 GOSUB 2840
1070 '
1080 ' ** SPREADSHEET CONTROL INPUT **
1090 '
1100 A1$=INKEY$ : IF A1$="" THEN GOTO 1100
1110 IF ASC(A1$)=27 THEN RETURN
1120 IF LEN(A1$)=2 THEN A2=ASC(RIGHT$(A1$,1)): GOSUB 1150 ELSE GOSUB 1330
1130 GOTO 1050
1140 '
1150 ' ** CURSOR KEYS **
1160 '
1170 IF A2=59 THEN GOSUB 1880: RETURN
1180 IF A2=80 OR A2=72 THEN GOSUB 2390: RETURN
1190 IF A2=77 AND NCOL<MXCOL THEN GOSUB 2950: NCOL=NCOL+NLEN: RETURN
ELSE IF A2=77 THEN RETURN
1200 IF A2=75 AND NCOL>MNCOL THEN GOSUB 2950: NCOL=NCOL-NLEN: RETURN
ELSE IF A2=75 THEN RETURN
1210 '
1220 ' ** PgUp..PgDn..Home..End **
1230 '
1240 IF A2=81 AND NFRST<=NCUR-15 THEN NFRST=NFRST+15 ELSE IF A2=81
THEN NFRST=NCUR
1250 IF A2=73 AND NFRST>15 THEN NFRST=NFRST-15 ELSE IF A2=73 THEN NFRST=1
1260 IF A2=71 THEN NFRST=1: 'HOME
1270 IF A2=79 THEN NFRST=NCUR: 'END
1280 IF A2=81 OR A2=73 OR A2=71 OR A2=79 THEN GOSUB 2950: COLOR 7,0:
GOSUB 2140: RETURN
1290 ' INPUT ERROR
1300 MSG$="INVALID KEY ENTRY":SCREEN 0,0,1,1

```

(continued)

```

1310 GOSUB 26000: SCREEN 0,0,0,0: LOCATE 24,1: PRINT BLANK$;: LOCATE 24,4:
PRINT "COMMAND = "; :RETURN
1320 '
1330 ' ** ALTER NUMERIC VALUE **
1340 '
1350 X=24: Y=15: GOSUB 21090 ' GET NUMERIC VALUE
1360 IF IERR=1 THEN RETURN
1370 NR=NFRST+NROW-8
1380 IF NR>NLMT THEN RETURN
1390 ON K GOSUB 1450,1490,1530
1400 IF NR>NCUR THEN NCUR=NCUR+1: LOCATE 3,53: PRINT USING "###";NCUR
1410 LOCATE NROW,1: I=NR: GOSUB 2640
1420 IF NR=NCUR AND NROW<21 THEN I=NCUR+1: LOCATE NROW+1,1: GOSUB 2640
1430 IF NCOL<MNCOL THEN GOSUB 2970: NCOL=NCOL+NLEN: RETURN
ELSE IF NROW<=21 THEN GOSUB 2970: NCOL=MNCOL:
NROW=NROW+1:RETURN
1440 RETURN
1450 ' VALUE ALTERATION NODES
1460 NC= INT(NCOL/10)+1
1470 IF NC=2 THEN XNOD(NR)=VALNEW: ELSE IF NC=3 THEN YNOD(NR)=VALNEW
1480 RETURN
1490 ' VALUE ALTERATION ELEMENTS
1500 NC= INT(NCOL/6)+1
1510 IF NC=2 THEN NCON(NR,1)=VALNEW: ELSE IF NC=3 THEN NCON(NR,2)=VALNEW:
ELSE IF NC=4 THEN AR(NR)=VALNEW
1520 RETURN
1530 ' LOAD VALUE ALTERATION
1540 NC= INT(NCOL/9)+1
1550 IF NC=1 THEN ILOAD(NR,1)=VALNEW: ELSE IF NC=2 THEN ILOAD(NR,2)=VALNEW:
ELSE IF NC=3 THEN RLOAD(NR)=VALNEW
1560 RETURN
1570 '
1580 ' PAGE HEADERS
1590 '
1600 CLS: LOCATE 25,5: PRINT FKEY$;
1610 LOCATE 1,(80-LEN(TITLE$))/2: PRINT TITLE$
1620 LOCATE 2,2:PRINT USING "PAGE = ##";PAGE
1630 IF K=1 THEN A1$="NODE" ELSE IF K=2 THEN A1$="ELEMENT" ELSE IF K=3
THEN A1$="LOAD"
1640 A1$=A1$+" EDITOR"
1650 LOCATE 2,(80-LEN(A1$)-8)/2: PRINT "... ";A1$;" ...": LOCATE 2,70:
PRINT DAT1$
1660 LOCATE 3,25: PRINT USING "CURRENT NUMBER OF ENTRIES = ###";NCUR
1670 ON K GOSUB 1690,1710,1750
1680 RETURN
1690 '** HEADER FOR NODES **
1700 PRINT : PRINT: PRINT " NODE X-COORD Y-COORD " : RETURN
1710 '** HEADER FOR ELEMENTS **
1720 PRINT
1730 PRINT " ELEM NODE NODE SECTION"
1740 PRINT " NO. I J AREA": RETURN
1750 '** HEADER FOR LOAD EDITING **
1760 PRINT:PRINT: PRINT " NODE I-CODE LOAD VALUE"
1770 ICOL=40
1780 LOCATE 8,ICOL: FOR I=ICOL TO 75: PRINT "*"; : NEXT I
1800 LOCATE 10,ICOL:PRINT "*** I-CODE .. MEANING ***"
1810 LOCATE 12,ICOL:PRINT "*** 0 FORCE IN X DIRECTION ***"
1820 LOCATE 13,ICOL:PRINT "*** 1 FORCE IN Y DIRECTION ***"
1830 LOCATE 14,ICOL:PRINT "*** 2 FIXITY IN X DIRECTION ***"
1840 LOCATE 15,ICOL:PRINT "*** 3 FIXITY IN Y DIRECTION ***"
1850 LOCATE 17,ICOL: FOR I=ICOL TO 75: PRINT "*"; : NEXT I
1860 RETURN
1870 '
1880 ' DELETE A LINE
1890 '
1900 LOCATE 24,1 : PRINT "TO DELETE THIS LINE PRESS. Y ..";
1910 IN$=INKEY$ : IF IN$="" THEN 1910
1920 IF IN$="Y" OR IN$="y" THEN GOSUB 2950: GOTO 1930 ELSE RETURN
1930 LOCATE 24,1 : PRINT BLANK$; : LOCATE 24,1 : PRINT "DELETION BEGINNING ";
1940 IF NFRST+NROW-8>NCUR THEN RETURN
1950 FOR I=NFRST+NROW-8 TO NCUR
1960 IF I=NLMT THEN 2080
1970 ON K GOTO 1980,2010,2040
1980 ' DELETE NODAL VALUES
1990 XNOD(I)=XNOD(I+1): YNOD(I)=YNOD(I+1)
2000 GOTO 2080
2010 ' DELETE ELEMENT VALUES

```



```

2020  AR(I)=AR(I+1): NCON(I,1)=NCON(I+1,1): NCON(I,2)=NCON(I+1,2)
2030  GOTO 2080
2040  ' BEGIN MODULE DELETE CURRENT LINE
2050      ILOAD(I,1)=ILOAD(I+1,1)
2060      ILOAD(I,2)=ILOAD(I+1,2)
2070      RLOAD(I)=RLOAD(I+1)
2080  NEXT I
2090  NCUR=NCUR-1
2100  COLOR 7,0: LOCATE 3,53: PRINT USING "###";NCUR
2110  GOSUB 2140
2120  RETURN
2130  '
2140  ' FULL DATA LIST
2150  '
2160  DEF SEG =&HB800: FOR I=1120 TO 3360 STEP 160: FOR J=0 TO (MXCOL+NLEN)*2
STEP 2: POKE I+J,0: NEXT J: NEXT I: DEF SEG
2170      LOCATE 8,1: NROW=8: IF NCUR<14 THEN NFRST=1
2180  ON K GOSUB 2200,2260,2330
2190  RETURN
2200  ' ** NODE POINTS **
2210  FOR I=NFRST TO NFRST+14
2220      IF I>NCUR+1 OR I>NLMT THEN 2240 ELSE IF I=NCUR+1 THEN
PRINT USING " ### " ;I: GOTO 2240
2230      PRINT USING " ### " ;I:: PRINT USING " ###.###"; XNOD(I),YNOD(I)
2240  NEXT I
2250  RETURN
2260  ' ** ELEMENTS **
2270  FOR I=NFRST TO NFRST+14
2280      IF I>NCUR+1 OR I>NLMT THEN 2310 ELSE IF I=NCUR+1 THEN
PRINT USING " ### " ;I: GOTO 2310
2290      PRINT USING " ### " ;I,NCON(I,1),NCON(I,2);
2300      PRINT USING "###.##";AR(I)
2310  NEXT I
2320  RETURN
2330  ' ** LOADS **
2340  FOR I=NFRST TO NFRST+14
2350      IF I>NCUR+1 OR I>NLMT THEN 2370
2360      PRINT USING " ### " ;ILOAD(I,1),ILOAD(I,2)::
PRINT USING "#####.###"; RLOAD(I)
2370  NEXT I
2380  RETURN
2390  '
2400  ' HANDLE UP/DOWN SCROLLING...AND MOVING CURSOR
2410  '
2420  IF A2=80 THEN 2530
2430  ' SCROLL DOWN
2440      IF NROW>8 THEN GOSUB 2940: NROW=NROW-1: RETURN
2450      IF NROW=8 AND NFRST=1 THEN RETURN ELSE NFRST=NFRST-1
2460  '
2470  ' SCROLL DOWN START AT 8 GOTO 23 ADD A LINE
2480  '
2490  DEF SEG =&HB800
2500  FOR I=3200 TO 1120 STEP -160: FOR J=0 TO (MXCOL+NLEN)*2 STEP 2:
I1=PEEK(I+J): POKE I+J+160,I1: NEXT J: NEXT I
2510  DEF SEG
2520  LOCATE 8,1: I=NFRST: GOSUB 2640: RETURN
2530  ' SCROLL UP
2540      IF NFRST+NROW-8=NCUR+1 THEN RETURN
2550      IF NROW<22 THEN GOSUB 2940: NROW=NROW+1: RETURN
2560      NFRST=NFRST+1
2570  '
2580  ' SCROLL UP START AT 8 GOTO 14 ADD A LINE
2590  '
2600  DEF SEG =&HB800
2610  FOR I=1280 TO 3360 STEP 160: FOR J=0 TO (MXCOL+NLEN)*2 STEP 2:
I1=PEEK(I+J): POKE I+J-160,I1: NEXT J: NEXT I
2620  DEF SEG
2630  LOCATE 22,1: I=NFRST+14: GOSUB 2640: RETURN
2640  '
2650  ' LIST 1 LINE OF DATA AT .I.
2660  '
2670  ON K GOSUB 2680,2720,2770: RETURN
2680  ' ** EDIT NODES **
2690      IF I=NCUR+1 THEN PRINT USING " ### " ;I:
RETURN

```

(continued)

```

2700 PRINT USING " ### ";I;: PRINT USING " ###.###";
XNOD(I),YNOD(I)
2710 RETURN
2720 ' ** EDIT ELEMENTS **
2730 IF I=NCUR+1 THEN PRINT USING " ### ";I: RETURN
2740 PRINT USING " ### ";I,NCON(I,1),NCON(I,2);
2750 PRINT USING "###.###";AR(I)
2760 RETURN
2770 ' ** EDIT LOADS **
2780 PRINT USING " ### ";ILOAD (I,1),ILOAD(I,2);:
PRINT USING "#####.###"; RLOAD(I)
2790 RETURN
2800 '
2810 ' HIGHLIGHT AN AREA ON THE SCREEN AT CURRENT
2820 ' CURSOR LOCATION (NROW,NCOL) LENGTH =NLEN
2830 '
2840 COLOR 0,7
2850 F$=""
2860 FOR L=1 TO NLEN
2870 LOCATE NROW,NCOL+L
2880 C$=CHR$(SCREEN(NROW,NCOL+L))
2890 PRINT C$
2900 NEXT L
2910 COLOR 7,0
2920 RETURN
2930 '
2940 ' REMOVE HIGHLIGHT ON THE SCREEN AT CURRENT
2950 ' CURSOR LOCATION (NROW,NCOL) LNEGTH=NLEN
2960 '
2970 COLOR 7,0
2980 F$=""
2990 FOR L=1 TO NLEN
3000 LOCATE NROW,NCOL+L
3010 C$=CHR$(SCREEN(NROW,NCOL+L))
3020 PRINT C$
3030 NEXT L
3040 COLOR 0,7
3050 RETURN
3060 '
3070 ' CONTROL CARDS
3080 '
3090 X=3: Y=8
3100 CLS: LOCATE 1,25: PRINT "TRUSS CONTROL VARIABLES"
3110 LOCATE 15,5: PRINT " USE UP/DOWN ARROWS TO POSITION CURSOR AND ENTER "
3120 LOCATE 16,5: PRINT " YOUR VALUE. USE RETURN KEY TO TERMINATE THE ENTRY"
3130 LOCATE 18,5: PRINT " <Esc> RETURNS TO MENU"
3140 LOCATE 3,1: PRINT "TITLE: ";TITLE$
3150 LOCATE 5,1: PRINT "DATE : ";DAT1$
3160 LOCATE 7,1: PRINT "ELASTIC MODULUS (PSI) : ";EKEY(1)
3170 LOCATE 9,1: PRINT "DESIGN STRENGTH (PSI) : ";EKEY(2)
3180 '
3190 ' TRAP KEYS
3200 '
3210 LOCATE X,Y,1
3220 A1$=INKEY$: IF A1$="" THEN 3220 ELSE IF ASC(A1$)=27 THEN 3400
3230 IF LEN(A1$)<>2 THEN GOTO 3300
3240 A1=ASC(RIGHT$(A1$,1)): IF A1=72 THEN GOTO 3250 ELSE IF A1=80 THEN
GOTO 3270 ELSE GOTO 3210
3250 IF X>7 THEN X=CSRLIN-2: Y=26: ELSE IF X<=7 AND X>3 THEN X=CSRLIN-2: Y=8
3260 GOTO 3210
3270 IF X>=5 AND X<9 THEN X=CSRLIN+2: Y=26: ELSE IF X=3 THEN X=5: Y=8
3280 GOTO 3210
3290 '
3300 ' VALUE ENTERED
3310 '
3320 IF ASC(A1$)<=32 OR ASC(A1$)>127 THEN GOTO 3140
3330 PRINT A1$; :X=CSRLIN
3340 IF CSRLIN=3 THEN LINE INPUT;TITLE$ :TITLE$=A1$+TITLE$
3350 IF CSRLIN=5 THEN LINE INPUT;DAT1$ :DAT1$=A1$+DAT1$
3360 IF CSRLIN=7 THEN LINE INPUT;A2$ :EKEY(1)=VAL(A1$+A2$)
3370 IF CSRLIN=9 THEN LINE INPUT;A2$ :EKEY(2)=VAL(A1$+A2$)
3380 IF X<=5 THEN Y=8 ELSE Y=26
3390 GOTO 3140
3400 '
3410 ' CHECK VALIDITY OF ENTRIES
3420 '
3430 IF EKEY(1)<1 OR EKEY(2)<1 THEN MSG$="INVALID PROPERTIES":

```



```

GOSUB 26000 :GOTO 3100
3440 IF LEN(DAT1$)<>10 THEN MSG$="INVALID DATE SPECIFICATION": GOSUB 26000 :
GOTO 3100
3450 RETURN
3460 '
3470 ' GRAPHICS
3480 '
3490 IMENU=3: GOSUB 25000: IOPT3=ISEL
3500 IF IRET<>0 THEN RETURN
3510 GOSUB 3540
3520 GOTO 3490
3530 '
3540 ' GRAPH MESH
3550 '
3555 IF IOPT3=1 THEN EXAG=0: GOTO 3570
3560 CLS: LOCATE 5,1
3563 PRINT " ENTER AN EXAGGERATION RATIO FOR THE NODE DEFLECTIONS"
3565 PRINT " IF YOU ENTER A ONE THEN STRUCTURE DISTORTION WILL BE "
3567 PRINT " TO SCALE...": INPUT EXAG
3568 IF EXAG<0 THEN EXAG=0
3570 CLS: LOCATE 12,30: PRINT "THINKING"
3575 '
3580 ' DETERMINE MAX/MIN
3585 '
3590 XMIN=XNOD(1)+EXAG*B(1): YMIN=YNOD(1)-EXAG*B(2): XMAX=XMIN: YMAX=YMIN
3600 FOR I=2 TO NUMNP
3610 IF IOPT3=1 THEN DXI=0: DYI=0 ELSE DXI=EXAG*B(I*2-1): DYI=+EXAG*B(I*2)
3620 IF XMIN>XNOD(I)+DXI THEN XMIN=XNOD(I)+DXI ELSE IF XMAX<XNOD(I)+DXI
THEN XMAX=XNOD(I)+DXI
3630 IF YMIN>YNOD(I)+DYI THEN YMIN=YNOD(I)+DYI ELSE IF YMAX<YNOD(I)+DYI
THEN YMAX=YNOD(I)+DYI
3640 NEXT I
3650 IF XMIN=XMAX OR YMIN=YMAX THEN MSG$="NODAL COORDINATE ERROR":
GOSUB 26000: RETURN
3652 '
3654 ' FIND MAXIMUM SCALE AND ALTER MAX/MIN TO FIT
3656 '
3660 S1=210/(XMAX-XMIN): S2=180/(YMAX-YMIN)
3670 IF S1>S2 THEN SM1=S2 ELSE SM1=S1
3675 DEL=.5*(210/SM1-XMAX+XMIN): XMAX=XMAX+DEL: XMIN=XMIN-DEL
3677 DEL=.5*(180/SM1-YMAX+YMIN): YMAX=YMAX+DEL: YMIN=YMIN-DEL
3679 '
3680 ' PLOT FROM (200,10)-(620,190)
3681 '
3690 PX=(XMAX-XMIN)/420
3700 PY=(YMAX-YMIN)/180
3710 SCREEN 2 :CLS
3720 FOR I1=1 TO NUMEL
3730 I=NCON(I1,1): J=NCON(I1,2)
3750 IF IOPT3=1 THEN DXI=0: DYI=0: DXJ=0: DYJ=0 ELSE
DXI=EXAG*B(I*2-1):DYI=+EXAG*B(I*2): DXJ=EXAG*B(J*2-1):DYJ=+EXAG*B(J*2)
3770 IX=INT(200+(XNOD(I)+DXI-XMIN)/PX)
3775 IY=INT(190-(YNOD(I)+DYI-YMIN)/PY)
3780 JX=INT(200+(XNOD(J)+DXJ-XMIN)/PX)
3782 JY=INT(190-(YNOD(J)+DYJ-YMIN)/PY)
3784 '
3786 ' DETERMINE IF PTS IN WINDOW
3788 '
3790 JCASE=0
3792 IF IX>620 OR IX<200 OR IY>190 OR IY<10 THEN JCASE=1
3794 IF JX>620 OR JX<200 OR JY>190 OR JY<10 THEN JCASE=JCASE+2
3796 '
3798 ON JCASE GOTO 3814,3814,3868
3800 LINE (IX,IY)-(JX,JY)
3802 PUT (IX-3,IY-3),NMK,OR: PUT (JX-3,JY-3),NMK,OR
3804 IF IOPT3=3 AND ABS(STR(I1))>=EKEY(2) THEN
LINE ((IX+JX)/2-4,(IY+JY)/2-2)-((IX+JX)/2+4,(IY+JY)/2+2),1,BF
3806 GOTO 3868
3808 '
3810 ' ONE POINT IN.. MAKE IT (IX,IY) OTHER IS (JX,JY)
3812 '
3814 IF JCASE=1 THEN KX=IX: KY=IY: IX=JX: IY=JY: JX=KX: JY=KY
3816 IF IX=JX AND JY>190 THEN JY=190: GOTO 3860 ELSE
IF IX=JX THEN JY=10: GOTO 3860
3818 SLOPE=(JY-IY)/(JX-IX): BINT=JY-SLOPE*JX
3820 '

```

(continued)

```

3822 IF SLOPE=0 AND JX>620 THEN JX=620: GOTO 3860 ELSE
IF SLOPE=0 THEN JX=200: GOTO 3860
3824 '
3826 KY=SLOPE*200+BINT
3828 IF KY>=10 AND KY<=190 AND SLOPE>0 AND KY>JY THEN JX=200: JY=KY:
GOTO 3860 ELSE IF KY>=10 AND KY<=190 AND SLOPE<0 AND KY<JY THEN
JX=200: JY=KY: GOTO 3860
3830 KY=SLOPE*620+BINT
3832 IF KY>=10 AND KY<=190 AND SLOPE>0 AND KY<JY THEN JX=620: JY=KY:
GOTO 3860 ELSE IF KY>=10 AND KY<=190 AND SLOPE<0 AND KY>JY THEN
JX=620: JY=KY: GOTO 3860
3834 '
3836 KX=(190-BINT)/SLOPE
3838 IF KX>=200 AND KX<=620 AND SLOPE>0 AND KX<JX THEN JX=KX: JY=190:
GOTO 3860 ELSE IF KX>=200 AND KX<=620 AND SLOPE<0 AND KX>JX THEN
JX=KX: JY=190: GOTO 3860
3840 KX=(10-BINT)/SLOPE
3842 IF KX>=200 AND KX<=620 AND SLOPE>0 AND KX>JX THEN JX=KX: JY=10:
GOTO 3860 ELSE IF KX>=200 AND KX<=620 AND SLOPE<0 AND KX<JX THEN
JX=KX: JY=10: GOTO 3860
3844 '
3860 LINE (IX,IY)-(JX,JY)
3862 PUT (IX-3,IY-3),NMK,OR
3868 NEXT I1
3870 '
3880 ' GRAPHICS MENU
3890 '
3900 LOCATE 1,1,1: SOUND 300,3: PRINT "SELECTION = "
3904 LOCATE 10,2: PRINT "1..NUMBER NODES"
3906 LOCATE 12,2: PRINT "2..NUMBER ELEMENTS"
3908 LOCATE 14,2: PRINT "3..ZOOM"
3910 LOCATE 16,2: PRINT "<Esc>..NEXT MENU"
3914 X=1: Y=12: GOSUB 21010: IF IERR=1 THEN SCREEN 0: RETURN
3916 IF VALNEW<1 OR VALNEW>4 THEN GOTO 3900 ELSE GOPT=VALNEW
3918 LOCATE 1,1: PRINT " THINKING "
3920 XX=EXAG
3930 ON GOPT GOSUB 3970,4050,4500
3940 IF GOPT=3 THEN GOTO 3660 ELSE GOTO 3900
3950 RETURN
3960 '
3970 ' NUMBER NODES
3980 '
3990 FOR I=1 TO NUMNP
4000 SX=200+(XNOD(I)-XMIN+XX*B(2*I-1))/PX:
SY=190-(YNOD(I)-YMIN+XX*B(2*I))/PY
4010 GOSUB 4170
4020 NEXT I
4030 RETURN
4040 '
4050 ' NUMBER ELEMENTS
4060 '
4070 FOR I=1 TO NUMEL
4080 II=NCON(I,1): JJ=NCON(I,2)
4090 XCOORD=(XNOD(II)+XNOD(JJ)+XX*(B(2*II-1)+B(2*JJ-1)))*.5
4100 YCOORD=(YNOD(II)+YNOD(JJ)+XX*(B(2*II)+B(2*JJ)))*.5
4110 SX=200+(XCOORD-XMIN)/PX: SY=190-(YCOORD-YMIN)/PY
4120 GOSUB 4170
4130 NEXT I
4140 RETURN
4150 '
4160 ' PUT A VALUE OF "I" AT (SX,SY) ON GRAPHICS SCREEN
4170 '
4180 IF SX<200 OR SX>620 OR SY<10 OR SY>190 THEN RETURN
4190 FDIG=INT(I/100)
4200 SDIG=INT((I-FDIG*100)/10)
4210 TDIG=INT(I-FDIG*100-SDIG*10)
4220 IF FDIG<=0 THEN 4270
4230 FOR II=0 TO 9
4240 IF FDIG=II THEN FOR J=0 TO 5: NDIG(J)=NUM(FDIG+1,J): NEXT J
4250 NEXT II
4260 PUT(SX-7,SY),NDIG,PSET
4270 IF FDIG<=0 AND SDIG<=0 THEN GOTO 4320
4280 FOR II=0 TO 9
4290 IF SDIG=II THEN FOR J=0 TO 5: NDIG(J)=NUM(SDIG+1,J): NEXT J
4300 NEXT II
4310 PUT(SX,SY),NDIG,PSET

```



```

4320 IF FDIG<=0 AND SDIG<=0 AND TDIG<=0 THEN RETURN
4330 FOR II=0 TO 9
4340 IF TDIG=II THEN FOR J=0 TO 5 : NDIG(J)=NUM(TDIG+1,J): NEXT J
4350 NEXT II
4360 PUT(SX+7,SY),NDIG,PSET
4370 RETURN
4495 '
4500 ' ZOOM FUNCTION.
4505 '
4510 LOCATE 25,1:PRINT BLANK$;: LOCATE 25,1: PRINT "LOCATE CROSSHAIRS TO ONE
CORNER OF ZOOM AREA AND PRESS RETURN KEY TO ENTER";
4520 XFIN=320:YFIN=100
4530 IF XFIN>630 THEN XFIN=630 ELSE IF XFIN<200 THEN XFIN=200 ELSE IF
YFIN>190 THEN YFIN=190 ELSE IF YFIN<10 THEN YFIN=10
4540 PUT (XFIN,YFIN),MARKER
4550 A1$=INKEY$: IF A1$="" THEN 4550 ELSE IF ASC(A1$)=27 THEN: RETURN
ELSE IF ASC(A1$)=13 THEN GOTO 4590 ELSE A1=ASC(RIGHT$(A1$,1))
4560 PUT (XFIN,YFIN),MARKER
4570 IF A1=77 THEN XFIN=XFIN+6 ELSE IF A1=75 THEN XFIN=XFIN-6 ELSE IF A1=72
THEN YFIN=YFIN-2 ELSE IF A1=80 THEN YFIN=YFIN+2
4580 GOTO 4530
4590 LOCATE 25,1:PRINT BLANK$;: LOCATE 25,1: PRINT "LOCATE CROSSHAIRS TO
OTHER CORNER OF ZOOM AREA AND PRESS RETURN KEY TO ENTER";
4600 XLOC1=XFIN:YLOC1=YFIN
4610 IF XFIN>630 THEN XFIN=630 ELSE IF XFIN<0 THEN XFIN=0 ELSE IF YFIN>199
THEN YFIN=199 ELSE IF YFIN<10 THEN YFIN=10
4620 PUT (XFIN,YFIN),MARKER
4630 A1$=INKEY$: IF A1$="" THEN 4630 ELSE IF ASC(A1$)=27 THEN RETURN
ELSE IF ASC(A1$)=13 THEN GOTO 4670 ELSE A1=ASC(RIGHT$(A1$,1))
4640 PUT (XFIN,YFIN),MARKER
4650 IF A1=77 THEN XFIN=XFIN+6 ELSE IF A1=75 THEN XFIN=XFIN-6 ELSE IF A1=72
THEN YFIN=YFIN-2 ELSE IF A1=80 THEN YFIN=YFIN+2
4660 GOTO 4610
4670 REM ** COMPUTE NEW SCALE FOR DWG. **
4680 IF (XLOC1-XFIN)=0 OR (YLOC1-YFIN)=0 THEN
MSG$="ZOOM POINTS IN HORIZONTAL OR VERTICAL LINE": GOSUB 26000:RETURN
4690 IF XLOC1<XFIN THEN XMAX=XMIN+PX*(XFIN-200): XMIN=XMIN+PX*(XLOC1-200)
ELSE XMAX=XMIN+PX*(XLOC1-200): XMIN=XMIN+PX*(XFIN-200)
4700 IF YLOC1<YFIN THEN YMAX=YMIN+PY*(190-YLOC1): YMIN=YMIN+PY*(190-YFIN)
ELSE YMAX=YMIN+PY*(190-YFIN): YMIN=YMIN+PY*(190-YLOC1)
4710 RETURN
5600 '
5610 ' GET/SAVE/DELETE DATA FILES
5620 '
5630 IMENU=4: GOSUB 25000: IOPT5=ISEL
5640 IF IRET<>0 THEN RETURN
5650 GOSUB 26400
5660 GOSUB 23000
5670 IF IERR=53 AND IOPT5=2 THEN GOTO 5680 ELSE IF IERR<>0 THEN GOTO 5630
5680 LOCATE 24,10
5690 IF IOPT5=1 THEN PRINT "ENTER FILE TO GET ";: ELSE IF IOPT5=2 THEN
PRINT "ENTER FILE TO SAVE";: ELSE PRINT "ENTER FILE TO DELETE";
5700 X=24: Y=33: GOSUB 21000
5710 IF IERR<>0 THEN GOTO 5630
5720 NAM$=INPT$
5730 ON IOPT5 GOSUB 5760,5940,6110
5740 GOTO 5630
5750 '
5760 ' LOAD AN EXISTING MESH
5770 '
5780 NAM$=DV$+NAM$: OPEN NAM$ FOR INPUT AS 3
5790 INPUT #3,TITLE$
5800 INPUT #3,NUMEL,NUMNP,NMAT,NLOAD
5810 INPUT #3,EKEY(1),EKEY(2),EKEY(3),EKEY(4),EKEY(5)
5820 FOR I=1 TO NUMNP
5830 INPUT #3,K,XNOD(K),YNOD(K)
5840 NEXT I
5850 FOR I=1 TO NUMEL
5860 INPUT #3, J,NCON(J,1),NCON(J,2),AR(J)
5870 NEXT I
5880 FOR I=1 TO NLOAD
5890 INPUT #3, ILOAD(I,1),ILOAD(I,2),RLOAD(I)
5900 NEXT I
5910 CLOSE #3: RETURN
5920 REM ** END MODULE TO RELOAD MATERIAL/LOAD DATA **
5930 '

```

(continued)

```

5940 ' SAVE MODEL DATA
5950 '
5960 NAM$=DV$+NAM$: OPEN NAM$ FOR OUTPUT AS 3
5970   WRITE #3,TITLE$
5980   WRITE #3,NUMEL,NUMNP,NMAT,NLOAD
5990   WRITE #3,EKEY(1),EKEY(2),EKEY(3),EKEY(4),EKEY(5)
6000   FOR I=1 TO NUMNP
6010     WRITE #3,I,XNOD(I),YNOD(I)
6020   NEXT I
6030   FOR I=1 TO NUMEL
6040     WRITE #3, I,NCON(I,1),NCON(I,2),AR(I)
6050   NEXT I
6060   FOR I=1 TO NLOAD
6070     WRITE #3, ILOAD(I,1),ILOAD(I,2),RLOAD(I)
6080   NEXT I
6090 CLOSE #3:RETURN
6100 '
6110 ' DELETE A FILE
6120 '
6130 CLS: SOUND 500,4
6140 LOCATE 5,30: PRINT " ***** WARNING *****"
6150   LOCATE 5,10: PRINT "A DELETED FILE CAN NOT BE RECOVERED ... TO DELETE"
6160   LOCATE 7,10: PRINT "   PRESS .. Y .. ELSE ANY OTHER KEY TO ABORT"
6170   A1$=INKEY$: IF A1$="" THEN 6170 ELSE IF A1$="Y" OR A1$="y" THEN
KILL DV$+NAM$: RETURN ELSE RETURN
10000 '
10010 ' ANALYSIS
10020 '
10070 IERR=0: CLS: LOCATE 2,30: PRINT "CHECKING DATA": PRINT
10074   FOR I=1 TO NUMEL
10078     IF NCON(I,1)=NCON(I,2) THEN PRINT "ELEMENT "+STR$(I)+" NODE ERROR":
IERR=1
10082   IF AR(I)<=0 THEN PRINT "ELEMENT "+STR$(I)+" AREA ERROR": IERR=1
10086   IF NCON(I,1)>NUMNP OR NCON(I,1)<=0 THEN PRINT "ELEMENT "+STR$(I)+
" NODE NOT DEFINED" : IERR=1
10087   IF NCON(I,2)>NUMNP OR NCON(I,2)<=0 THEN PRINT "ELEMENT "+STR$(I)+
" NODE NOT DEFINED" : IERR=1
10088   NEXT I
10090   LOCATE 25,10: PRINT "PRESS ANY KEY TO CONTINUE";
10094   A1$=INKEY$: IF A1$="" THEN 10094
10096   IF IERR<>0 OR ASC(A1$)=27 THEN IERR=0: RETURN
10100   GOSUB 30000
10102   IF BW>MBW THEN MSG$="BANDWIDTH EXCEEDS ALLOWABLE":GOSUB 26000: RETURN
10105   IF NUMNP>MAXNP THEN MSG$="TOO MANY NODES": GOSUB 26000: RETURN
10107   NDF=NUMNP*2
10108   FOR II=1 TO NDF
10110     FOR JJ=1 TO BW
10120       A(II,JJ)=0: B(II)=0
10130     NEXT JJ
10140   NEXT II
10150 '
10160 ' COMPUTE ELEMENT STIFFNESS/ADD TO GLOBAL
10170 '
10180   FOR IELE=1 TO NUMEL
10190     GOSUB 10700
10200   NEXT IELE
10210 '
10220 ' IMPOSE LOADS/FIXITIES
10230 '
10240   CP=10^12
10250   FOR I=1 TO NLOAD
10260     IF ILOAD(I,1)>NUMNP THEN MSG$="ERROR IN LOAD"+STR$(I): GOSUB
26000:RETURN
10270     IF ILOAD(I,2)=0 OR ILOAD(I,2)=1 THEN J=2*ILOAD(I,1)-1+ILOAD(I,2):
B(J)=RLOAD(I)
10280     IF ILOAD(I,2)=2 OR ILOAD(I,2)=3 THEN J=2*ILOAD(I,1)-3+ILOAD(I,2):
A(J,1)=A(J,1)*CP: B(J)=RLOAD(I)*A(J,1)
10290   NEXT I
10300 '
10310 ' SOLVE MATRIX
10320 '
10330   FOR N=1 TO NDF
10340     FOR L=2 TO BW
10350       IF A(N,L)=0 THEN GOTO 10450
10360       I=N+L-1
10370       C=A(N,L)/A(N,1)

```



```

10380 J=0
10390 FOR K=L TO BW
10400 J=J+1
10410 A(I,J)=A(I,J)-C*A(N,K)
10420 NEXT K
10430 A(N,L)=C
10440 B(I)=B(I)-A(N,L)*B(N)
10450 NEXT L
10460 B(N)=B(N)/A(N,1)
10470 NEXT N
10480 REM ** BACK SUBSTITUTION **
10490 FOR M=2 TO NDF
10500 N=NDF+1-M
10510 FOR L=2 TO BW
10520 IF A(N,L)=0 THEN GOTO 10550
10530 K=N+L-1
10540 B(N)=B(N)-A(N,L)*B(K)
10550 NEXT L
10560 NEXT M
10570 '
10580 ' CALCULATE STRESSES
10590 '
10600 IF NUMNP<=0 THEN RETURN
10610 FOR IELE=1 TO NUMEL
10620 NN1=NCON(IELE,1)
10630 NN2=NCON(IELE,2)
10640 X=XNOD(NN2)-XNOD(NN1)
10650 Y=YNOD(NN2)-YNOD(NN1)
10660 E=EKEY(1)/(X^2+Y^2)
10670 STR(IELE)=E*(X*(B(2*NN2-1)-B(2*NN1-1))+Y*(B(2*NN2)-B(2*NN1)))
10680 NEXT IELE
10690 RETURN
10700 '
10710 ' ELEMENT STIFFNESS... ELEMENT # IELE
10720 '
10730 NN1=NCON(IELE,1)
10740 NN2=NCON(IELE,2)
10750 X=XNOD(NN2)-XNOD(NN1)
10760 Y=YNOD(NN2)-YNOD(NN1)
10770 AEL=AR(IELE)*EKEY(1)/(SQR(X^2+Y^2))^3
10780 XXYY(1)=-X
10790 XXYY(2)=X
10800 XXYY(3)=-Y
10810 XXYY(4)=Y
10820 FOR I= 1 TO 4
10830 FOR J=1 TO 4
10840 BK(I,J)=AEL*XXYY(I)*XXYY(J)
10850 NEXT J
10860 NEXT I
10870 FOR I1=1 TO 2
10880 FOR I2=1 TO 2
10890 FOR J1=1 TO 2
10900 FOR J2=1 TO 2
10910 JJ=2*NCON(IELE,J2)-2+J1
10920 II=2*NCON(IELE,I2)-1+I1-JJ
10930 IF II>0 THEN
I=2*(I1-1)+I2: J=2*(J1-1)+J2:
A(JJ,II)=A(JJ,II)+BK(I,J)
10940 NEXT J2
10950 NEXT J1
10960 NEXT I2
10970 NEXT I1
10980 RETURN
20000 '
20010 ' ROUTINE: SCREEN RESET AND CW NOTICE
20020 '
20030 KEY OFF
20040 FOR I=1 TO 10: KEY I,"" : NEXT I
20050 FOR J=0 TO 3: SCREEN 0,0,J,0: CLS: NEXT J
20060 '
20070 ' PRINT COPYRIGHT NOTICE
20080 '
20090 SCREEN 0,0,0,1
20100 FOR I=5 TO 70 STEP 5: LOCATE 6,I: PRINT "-----"; : LOCATE 20,I:
PRINT "-----"; : NEXT I

```

(continued)

July

```
20110 LOCATE 9,36: PRINT "TRUSS2";
20120 LOCATE 11,15: PRINT "REVISION 1.00 .... COPYRIGHT .. C.S. PEDICINI
1984"
20130 COLOR 0,15
20140 LOCATE 16,25,0 : PRINT " PRESS ANY KEY TO CONTINUE "
20150 COLOR 7,0
20160 SCREEN 0,0,0,0
20170 A1$=INKEY$: IF A1$="" THEN 20170
20180 RETURN
21000 '
21010 ' STRING INPUT MODULE
21020 '   X,Y ... CURSOR LOCATION TO PRINT STRING
21030 '   A1$ ... MAY CONTAIN FIRST CHARACTER
21040 '   IERR... 1 FOR ABORT
21050 '   VALNEW. NUMERIC RESULT
21060 '   INPT$...STRING RESULT
21070 '
21080 A1$=""
21090 IERR=0 : INPT$=A1$: IF LEN(A1$)=0 THEN GOTO 21120
21100 IF ASC(A1$)=13 THEN 21190
21110 IF ASC(A1$)=27 THEN IERR=1: RETURN
21120 LOCATE X,Y: PRINT INPT$;
21130 A1$=INKEY$:IF A1$="" THEN 21130
21140 IF ASC(A1$)=8 AND LEN(INPT$)>=1 THEN INPT$=MID$(INPT$,1,LEN(INPT$)-1):
LOCATE X,Y: PRINT INPT$;" ";: GOTO 21130
21150 IF ASC(A1$)=27 THEN IERR=1 :RETURN
21160 IF ASC(A1$)=13 THEN 21190
21170 INPT$=INPT$+A1$:LOCATE X,Y: PRINT INPT$;: GOTO 21130
21180 '
21190 ' CONVERT VALUE
21200 '
21210 VALNEW=VAL(INPT$)
21220 RETURN
21230 '
22000 '
22010 ' ERROR TRAP MODULE
22020 '   ERR . BASIC ERROR CODE
22030 '   MSG$ . ERROR TEXT
22040 '
22080 MSG$=""
22090 IF ERR=24 OR ERR=25 OR ERR=27 THEN MSG$="PRINTER ERROR"
22100 IF ERR=53 THEN MSG$="FILE DOES NOT EXIST"
22110 IF ERR=57 THEN MSG$="DOS ERROR ON DISKETTE I/O"
22120 IF ERR=61 THEN MSG$="DISK FULL"
22130 IF ERR=62 THEN MSG$="INPUT PAST END OF FILE"
22140 IF ERR=64 THEN MSG$="BAD FILE NAME"
22150 IF ERR=70 THEN MSG$="DISK WRITE PROTECTED"
22160 IF ERR=71 THEN MSG$="DISK NOT READY"
22170 IF ERR=72 THEN MSG$="DISK MEDIA ERROR"
22180 IF MSG$="" THEN MSG$="ERROR CODE "+STR$(ERR)
22190 '
22200 ' PRINT ERROR MSG
22210 '
22220 IERR=ERR: SCREEN 0,0,1,1: GOSUB 26000: SCREEN 0,0,0,0
22230 RESUME 22240
22240 RETURN
23000 '
23010 ' DETERMINES DIRECTORY FOR PATH DV$ USING SCREENS 0,2 AND 3
23020 ' FILE LIST IS IN LINES 3-22..LINE 24 IS COMMAND LINE
23030 '   DV$..... DIRECTORY/PATHNAME
23040 '   NUMFIL.. NUMBER OF FILES
23050 '   I,J,K... USED AS INDICES
23060 '   X,Y..... LOCATION OF FILE FOR SCREEN 0
23070 '
23090 CLS: NAM$="": IERR=0: SCREEN 0,0,2,2: CLS
23100 LOCATE 12,35: COLOR 0,15: PRINT "SORTING":COLOR 7,0
23110 '
23120 ' XFER FILES FROM SCREEN 3 TO SCREEN 1... FIND # OF FILES
23130 '
23135 I1=2: I2=18: I3=17: I4=I3-11: I5=4 ' INTERPRETIVE MODE
23137 ' I1=1: I2=13: I3=12: I4=I3-11: I5=6 ' COMPILED MODE
23140 SCREEN 0,0,3,2: CLS: FILES DV$+"*.*": NUMFIL=0
23150 FOR I=I1 TO 24
23160 FOR J=1 TO I5
23170 SCREEN 0,0,3,2
23180 IF SCREEN(I,I2*J-I3)=32 OR SCREEN(I,I2*J-I3)=0 THEN GOTO 23260
```



```

23190 A1$=""
23200 FOR K=-I3 TO -I4
23210 A1$=A1$+CHR$(SCREEN(I,I2*J+K))
23220 NEXT K
23230 NUMFIL=NUMFIL+1
23240 X=15*INT(NUMFIL/20)+1: Y=NUMFIL-INT(X/15)*19+2
23250 SCREEN 0,0,0,2: LOCATE Y,X: PRINT A1$
23260 NEXT J
23270 NEXT I
23280 SCREEN 0,0,0
23290 LOCATE 1,30: PRINT " DRIVE CODE = ";DV$
23300 RETURN
23310 '
23320 ' ERROR TRAP
23330 '
23340 GOSUB 22000
23350 IF ERR=53 THEN RESUME 23280
23360 GOSUB 26000: IERR=1
23370 RETURN
24000 '
24010 ' TRANSFER CONTROL
24020 '
24030 CLS
24040 CLS: SOUND 400,2: LOCATE 6,30
24050 PRINT " ***** WARNING *****":LOCATE 7,20
24060 PRINT " BEFORE YOU EXIT THIS PROGRAM MAKE SURE YOU":LOCATE 8,23
24070 PRINT " HAVE SAVED ALL OF YOUR DATA FILE(S) "
24080 LOCATE 13,30: PRINT "ARE YOU SURE Y=YES"
24090 A1$=INKEY$:IF A1$="" THEN 24090 ELSE IF A1$="y" OR A1$="Y" THEN SYSTEM
24100 RETURN
25000 '
25010 ' MENU PRINT/ENTRY
25020 ' MENU$(10,6)... 10 MENUS, 6 ITEMS PER MENU
25030 ' IMENU..... MENU TO BE PRINTED
25040 ' ISEL..... ITEM SELECTED FROM MENU
25050 ' IRET..... <>0 IF ABORTED
25060 '
25070 CLS
25080 LOCATE 1,(80-LEN(TITLE$))/2: PRINT TITLE$
25090 LOCATE 1,70: PRINT DAT1$
25100 LOCATE 4,30: PRINT MENU$(IMENU,1)
25110 '
25120 OPTMAX=0
25130 FOR I=2 TO 7
25140 IF MENU$(IMENU,I)="" THEN 25170
25150 OPTMAX=OPTMAX+1
25160 LOCATE 2*I+3,27: PRINT I-1;" ";MENU$(IMENU,I)
25170 NEXT I
25180 '
25190 ' ACCEPT OPTION
25200 '
25210 LOCATE 25,60: PRINT "ESC = MAIN MENU";
25220 IRET=0
25230 LOCATE 22,30: COLOR 0,15: PRINT "SELECTION = "; :COLOR 7,0
25240 X=22: Y=43: GOSUB 21000
25250 IF IERR=1 THEN IRET=1: RETURN
25260 IF VAL(INPT$)<>0 AND VAL(INPT$)<=OPTMAX THEN ISEL=VAL(INPT$): RETURN
25270 MSG$="INVALID MENU RESPONSE ": SCREEN 0,0,1,1: GOSUB 26000:
SCREEN 0,0,0,0: INPT$="": LOCATE 22,42: PRINT " " " :GOTO 25240
26000 '
26010 ' ERROR SUBROUTINE
26020 ' MSG$... CONTAINS ERROR MESSAGE
26030 '
26040 CLS: SOUND 600,6
26050 LN=LEN(MSG$)
26060 LOCATE 7,32: PRINT " P R O B L E M "
26070 LOCATE 10,INT((80-LN)/2): PRINT MSG$
26080 LOCATE 15,27: PRINT "PRESS ANY KEY TO CONTINUE";
26090 A1$=INKEY$: IF A1$="" THEN 26090 ELSE RETURN
26100 '
26400 '
26410 ' CURRENT DIRECTORY/PATH
26420 '
26430 CLS
26440 LOCATE 12,10
26450 PRINT "CURRENT DIRECTORY: ";; COLOR 0,7: PRINT " ";DV$;" " :COLOR 7,0

```

(continued)


```

26460 LOCATE 14,10
26470 PRINT "NEW DIRECTORY:"
26472 LOCATE 20,10: PRINT " TO CHANGE DIRECTORIES/DRIVES ENTER THE NAME OF "
26474 LOCATE 21,10: PRINT " THE NEW DRIVE AND PRESS THE RETURN KEY. TO
RETAIN
26476 LOCATE 22,10: PRINT " THE CURRENT ONE PRESS THE RETURN KEY."
26478 LOCATE 14,25,1
26480 A1$=INKEY$: IF A1$="" THEN 26480 ELSE IF ASC(A1$)=13 THEN RETURN
26490 X=14: Y=25: COLOR 0,7: GOSUB 21090: COLOR 7,0
26500 IF IERR<>0 THEN RETURN
26510 IF RIGHT$(INPT$,1)=":" THEN DV$=INPT$: RETURN
26520 MSG$="INVALID DRIVE SPECIFICATION": GOSUB 26000: GOTO 26400
30000 '
30010 ' DTERMINE BANDWIDTH ..... REF SEGERLIND PG 18
30020 ' NCON(I,J)... ELEMENT NUMBERS
30030 ' MDIFF..... MAX NODE DIFFERENCE IN AN ELEMENT
30040 ' NUMEL..... NUMBER OF ELEMENTS
30050 ' BW..... BANDWIDTH
30060 '
30070 MDIFF=0: CLS: LOCATE 12,35: PRINT "WORKING"
30080 FOR I=1 TO NUMEL
30090 DIFF=ABS(NCON(I,1)-NCON(I,2)): IF DIFF>MDIFF THEN MDIFF=DIFF
30100 NEXT I
30110 BW=2*(MDIFF+1)
30120 RETURN
30130 '
31000 '
31010 ' ROUTINE TO PRINT TO SCREEN OR LINEPRINTER
31020 ' PRNT$(5)..... ARRAY OF Y/N PRINT CHOICES
31030 '
31040 CLS: LOCATE 1,28: PRINT "PRINTER CONTROL VARIABLES"
31050 LOCATE 5,28: PRINT "ENTER TO PRINT TO SCREEN"
31060 LOCATE 7,28: PRINT "ENTER TO PRINT LINEPRINTER"
31070 COLOR 0,15: LOCATE 5,34: PRINT " S ": LOCATE 7,34: PRINT " L ":
COLOR 7,0
31080 LOCATE 13,30: PRINT "ENTRY =": X=13: Y=38: GOSUB 21000
31090 IF INPT$="S" OR INPT$="s" THEN INPT$="S": GOTO 31120 ELSE
IF INPT$="L" OR INPT$="l" THEN INPT$="L": GOTO 31120
31100 RETURN
31110 '
31120 ' ESTABLISH PRINT OPTIONS
31130 '
31140 X=5: Y=28
31150 PRNT$(1)="Y": FOR I=2 TO 5: PRNT$(I)=PRNT$(1): NEXT I
31160 CLS
31170 LOCATE 17,5: PRINT " USE UP/DOWN ARROWS TO POSITION CURSOR AND ENTER "
31180 LOCATE 18,5: PRINT " Y/N ..CAPITALS ONLY!.. USE F10 TO CONTINUE"
31190 LOCATE 20,5: PRINT " <Esc> RETURNS TO MENU"
31200 LOCATE 3,1: PRINT " PRINT Y/N"
31210 LOCATE 5,1: PRINT "NODE POINT DATA..... ";PRNT$(1)
31220 LOCATE 7,1: PRINT "ELEMENT DATA..... ";PRNT$(2)
31230 LOCATE 9,1: PRINT "LOAD AND MATERIAL DATA.... ";PRNT$(3)
31240 LOCATE 11,1:PRINT "STRESS/STRAIN RESULTS..... ";PRNT$(4)
31250 LOCATE 13,1:PRINT "NODE DISPLACEMENTS..... ";PRNT$(5)
31260 '
31270 ' TRAP KEYS
31280 '
31290 LOCATE X,Y,1
31300 A1$=INKEY$: IF A1$="" THEN 31300 ELSE IF ASC(A1$)=27 THEN RETURN
31310 IF LEN(A1$)<>2 THEN IF A1$="Y" OR A1$="N" THEN LOCATE X,Y,1:
PRINT A1$: PRNT$(INT(X/2-1.5))=A1$: GOTO 31290 ELSE GOTO 31290
31320 A1=ASC(RIGHT$(A1$,1))
31330 IF A1=72 AND X>=7 THEN X=CSRLIN-2
31340 IF A1=80 AND X<13 THEN X=CSRLIN+2
31350 IF A1<>68 THEN 31290
31360 '
31370 ' PRINT DATA
31380 '
31382 GOSUB 31390: CLOSE #1: RETURN ' STANDARD ESCAPE
31390 IF INPT$="S" THEN OPEN "SCRN:" FOR OUTPUT AS #1: GOTO 31470
31400 OPEN "LPT1:" FOR OUTPUT AS #1
31410 CLS: LOCATE 10,10: PRINT "SET UP PRINTER AND PRESS ANY KEY WHEN READY
"
31420 A1$=INKEY$ : IF A1$="" THEN 31420 ELSE IF ASC(A1$)=27 THEN RETURN
31430 CLS: LOCATE 13,30:PRINT "PRINTING DATA";
31440 LOCATE 16,20: PRINT "TO STOP PRINTING PRESS ESC KEY"
31450 PRINT #1,CHR$(27)"N"CHR$(6)

```



```

31460 '
31470 ' OUTPUT NODE DATA
31480 '
31490 IF PRNT$(1)<>"Y" THEN 31610
31530 FOR I=1 TO NUMNP
31540 A1$=INKEY$: IF A1$=CHR$(27) THEN RETURN
31545 IF I=1 OR CSRLIN=23 THEN GOSUB 32190:
PRINT #1, TAB(10);"NODE X-COORD Y-COORD";CHR$(10);CHR$(10)
31550 PRINT #1, TAB(10);
31560 PRINT #1, USING " ### ";I;
31570 PRINT #1, USING " ###.####";XNOD(I),YNOD(I)
31590 NEXT I
31600 '
31610 ' LIST ELEMENTS
31620 '
31630 IF PRNT$(2)<>"Y" THEN 31760
31680 FOR I=1 TO NUMEL
31690 A1$=INKEY$: IF A1$=CHR$(27) THEN RETURN
31695 IF I=1 OR CSRLIN=23 THEN GOSUB 32190:
PRINT #1, TAB(10);"ELEM NODE NODE SECTION":
PRINT #1, TAB(10);" NO. I J AREA",CHR$(10)
31700 PRINT #1, TAB(10);
31710 PRINT #1, USING " ### ";I,NCON(I,1),NCON(I,2);
31720 PRINT #1, USING "###.###";AR(I)
31740 NEXT I
31750 '
31760 ' LIST LOADS
31770 '
31780 IF PRNT$(3)<>"Y" THEN 31890
31820 FOR I=1 TO NLOAD
31830 A1$=INKEY$: IF A1$=CHR$(27) THEN RETURN
31835 IF I=1 OR CSRLIN=23 THEN GOSUB 32190:
PRINT #1, TAB(10);"NODE NO. CODE VALUE",CHR$(10),CHR$(10)
31840 PRINT #1, TAB(10);
31850 PRINT #1, USING " ### ";ILOAD(I,1),ILOAD(I,2);
31860 PRINT #1, USING " #.##^";RLOAD(I)
31870 NEXT I
31880 '
31890 ' OUTPUT RESULTS
31900 '
31910 IF PRNT$(4)<>"Y" THEN 32020
31950 FOR I=1 TO NUMNP
31960 A1$=INKEY$: IF A1$=CHR$(27) THEN RETURN
31965 IF I=1 OR CSRLIN=23 THEN GOSUB 32190:
PRINT #1, TAB(10);"NODE NO X-DISP. Y-DISP. ",CHR$(10),CHR$(10)
31970 PRINT #1, TAB(10);
31980 PRINT #1, USING " ### ";I;
31990 PRINT #1, USING " ###.####";B(2*I-1),B(2*I)
32000 NEXT I
32010 '
32020 ' OUTPUT RESULTS
32030 '
32040 IF PRNT$(5)<>"Y" THEN 32170
32080 FOR I=1 TO NUMEL
32090 A1$=INKEY$: IF A1$=CHR$(27) THEN RETURN
32095 IF I=1 OR CSRLIN=23 THEN GOSUB 32190:
PRINT #1, TAB(10);"ELEM NO STRESS-PSI FORCE-LBS",CHR$(10),CHR$(10)
32100 PRINT #1, TAB(10);
32110 PRINT #1, USING " ### ";I;
32120 PRINT #1, USING " ###.####";STR(I),STR(I)/AR(I)
32130 NEXT I
32140 IF INPT$<>"S" THEN CLOSE #1: RETURN
32150 LOCATE 25,5: PRINT "PRESS ANY KEY TO CONTINUE";
32160 A1$=INKEY$: IF A1$="" THEN 32160 ELSE RETURN
32170 CLOSE #1: RETURN
32180 '
32190 ' PAGE HEADER
32200 '
32210 IF INPT$<>"S" THEN X=5: WHILE X: LPRINT CHR$(10): X=X-1: WEND: GOTO
32240
32220 LOCATE 25,5: PRINT "PRESS ANY KEY TO CONTINUE";
32230 A1$=INKEY$: IF A1$="" THEN 32230 ELSE CLS
32240 PRINT #1, CHR$(10)
32250 PRINT #1, SPC(1);DAT1$
32260 PRINT #1, TAB(40-LEN(TITLE$)/2); TITLE$
32270 PRINT #1, CHR$(10)
32280 RETURN

```

ughbug.asm

"An 8031 In-Circuit Emulator," by George Dinwiddie. July,
page 181.

```
TITLE ' UGHBUG                VERSION 1.00          12 January 1986 '
SBTTL ' Copyright 1986 by George Dinwiddie '
PGLEN  55
```

```
:
: NOTICE : This monitor program may be copied and distributed on the
: condition that all copyright notices and this notice remain intact.
: This program is provided without any warranty or assumption of
: liability whatsoever. If it doesn't work or it breaks something, I'm
: sorry, but it's not my problem.
:
```

```
: On the positive side, if this program helps you, I am glad to be of
: assistance. I ask only two favors. First, if you fix any bugs or
: add any enhancements, please send me a copy. (CP/M 8" SSSD or IBM-PC
: 5.25" are best for me.) Secondly, if you use this program to help
: develop a commercial product, please remember me. As a suggestion, a
: $50.00 donation would be a bargain for you and welcome by me. If you
: would prefer, either a sample of your product or a donation more in
: line with the profits on your product would be ok, too. If you use
: this program on a home project, I understand that you can't always
: afford to pay for the value of good software. My own CP/M computer
: would be worthless without the wonderful public domain software that
: I have received for free. If you want to send me your favorite
: public domain goody, great. In any event, you are welcome to this --
: enjoy, enjoy.
:
```

```
: George Dinwiddie
: 10965 Trotting Ridge Way
: Columbia, MD 21044
:
```

```
: REVISION HISTORY :
```

```
: Version 1.00 George Dinwiddie 12 January 1986.
: First public release. Fixed GO-from-break bug.
: Version 0.16 George Dinwiddie 26 January 1984
: Added JUMPTABLE and HEXMATH functions.
:
```

```
: WISH LIST :
```

```
: Load command ('L') to load ram from a hex file through the terminal
: port.
:
```

```
: Find command ('F') to search for sequences specified in hex or
: ascii. I always intended to add this one but never did. That's
: the reason the Insert command isn't a Fill command.
:
```

```
: Disassemble command (maybe 'U' for Unassemble).
:
```

```
: Single-line assembler. It's amazing how good you get at hand
: assembly when you practice. Still, it would be nice to have this.
:
```

```
: NOTE : Some labels are of the following form :
```

```
: IF000 == IF
: THN000 == THEN
: ELS000 == ELSE
: NDI000 == ENDIF
:
```

```
MONBASE EQU 0000H ;4K MONITOR
APPBASE EQU 2000H ;8K APPLICATION EPROM
RAMBASE EQU 4000H ;8K RAM
UARTBASE EQU 7800H ;TERMINAL (uart) LOCATION
UARTDATA EQU UARTBASE ; C/D* = A0
UARTCONT EQU (UARTBASE+1)
RAMEND EQU 5FFFH ;END OF EXTERNAL RAMSPACE
BYTENUM EQU (RAMEND-8) ;FROM HERE ON USED BY BREAK ROUTINE
STACK EQU 50H ;stack begins here
```



```
PCON          EQU      87H      ;AVOCET ASSEMBLER DOESN'T KNOW PCON

;reserved internal ram locations
RESERVED      EQU      48H      ;reserved internal ram
HIBYTE        EQU      RESERVED ;TOP OF 16-BIT ADDRESS
LOBYTE        EQU      RESERVED+1 ;BOTTOM OF 16 BIT ADDRESS
FIRST         EQU      RESERVED+2 ;START ADDRESS HIGH (LOW IN RESERVED+3)
LAST          EQU      RESERVED+4 ;END ADDRESS HIGH (LOW IN RESERVED+5)
TO            EQU      RESERVED+6 ;TO ADDRESS HIGH (LOW IN RESERVED+7)

;reserved register bank (and aliases for those registers)
RESBANK       EQU      1        ;register bank 1
REG0          EQU      8*RESBANK
REG1          EQU      REG0+1    ;sometimes you can't use R1
REG2          EQU      REG0+2
REG3          EQU      REG0+3
REG4          EQU      REG0+4
REG5          EQU      REG0+5
REG6          EQU      REG0+6
REG7          EQU      REG0+7

;character equates
CR            EQU      0DH       ;CARRIAGE RETURN
LF            EQU      0AH       ;LINE FEED
SPACE        EQU      20H       ;SPACE CHARACTER
TAB          EQU      09H       ;TAB CHARACTER
DOT          EQU      2EH       ;PERIOD
BACKSP       EQU      08H       ;BACKSPACE
EOT          EQU      04H       ;end of text
```

```
*****
```

```
START:  ORG      MONBASE
        JMP      UGHBUG

        ORG      START+03H
        LJMP     RAMBASE+03H      ;INTERRUPT VECTORS JUMP TO RAM
        ORG      START+0BH
        LJMP     RAMBASE+0BH
        ORG      START+13H
        LJMP     RAMBASE+13H
        ORG      START+1BH
        LJMP     RAMBASE+1BH
        ORG      START+23H
        LJMP     RAMBASE+23H
```

```
*****
```

```
LJMPTBL:  LJMP     COOL
          LJMP     WARM

          LJMP     IN
          LJMP     INCH
          LJMP     INHEX
          LJMP     BYTES
          LJMP     BADDR
          LJMP     THRADR

          LJMP     OUT
          LJMP     OUTCH
          LJMP     OUTS
          LJMP     OUT2S
          LJMP     CRLF
          LJMP     OUT2H
          LJMP     OUTR0
          LJMP     OUTC2HS
          LJMP     PDATA

          LJMP     INC16
          LJMP     DEC16
          LJMP     CPY
          LJMP     BRKPT
          PAGE

MSIGNON:  DB      CR, LF, 'Ughbug MCS-51 monitor, version 1.00'
```

(continued)

DB CR, LF, 'copyright 1986 by George Dinwiddie.', LF, 04H

```

UGHBUG: MOV    SP,#STACK
        MOV    STACK,SP      ;store stack pointer for cool start
        SETB   RS0           ;SELECT REGISTER BANK 1
        CLR    RS1
        CLR    A
        MOV    DPTR,#BYTENUM
        MOV    R0,#(RAMEND-BYTENUM+1)
CLEAR:  MOVX   @DPTR,A        ;clear space used by break routine
        INC    DPTR
        DJNZ   R0,CLEAR
DLAY:   DJNZ   ACC,DLAY        ;WAIT HERE FOR UART TO WAKE-UP
        DJNZ   R0,DLAY
        MOV    DPTR,#UARTCONT ;INIT UART
        CLR    A
        MOVX   @DPTR,A        ;send three zeros because you don't know
        NOP    ;in what crazy mode the ugh-art wakes up
        DJNZ   ACC,$-1
        MOVX   @DPTR,A
        NOP
        DJNZ   ACC,$-1
        MOVX   @DPTR,A
        NOP
        DJNZ   ACC,$-1
        MOV    A,#40H         ;software reset uart
        MOVX   @DPTR,A
        CLR    A              ;many thanks to Ernest Penzenstadler
        NOP    ;for taming the infamous 8251A ugh-art!
        DJNZ   ACC,$-1
        MOV    A,#01001110B   ;MODE
        MOVX   @DPTR,A
        MOVX   A,@DPTR        ;GET STATUS
        JNB    ACC.0,$-1      ;WAIT FOR TXRDY
        MOV    A,#00110111B   ;COMMAND
        MOVX   @DPTR,A
        MOV    DPTR,#MSIGNON
        CALL   PDATA
COOL:   MOV    SP,STACK

```

```

WARM:   MOV    DPTR,#MPROMPT  ;WARM START
        CALL   PDATA
        CALL   INCH
        JNB    ACC.6,$+5      ;JUMP IF NOT A LETTER
        CLR    ACC.5          ;CONVERT LOWER TO UPPER CASE
        MOV    R7,A           ;SAVE CHARACTER INPUT
        MOV    DPTR,#FUNTAB   ;POINT TO FUNCTION TABLE
SCAN:   CLR    A
        MOVC   A,@A+DPTR
        JZ     WARM           ;END OF TABLE
        CJNE   A,0FH,NEXT     ;COMPARE WITH SAVED CHARACTER
        MOV    A,#01H
        MOVC   A,@A+DPTR      ;GET HIGH BYTE OF JUMP
        MOV    R2,A           ;SAVE IN R2
        MOV    A,#02H
        MOVC   A,@A+DPTR      ;GET LOW BYTE OF JUMP
        MOV    R3,A           ;SAVE IN R3
        PUSH   REG3           ;PUSH R3 (LOW BYTE)
        PUSH   REG2           ;PUSH R2 (HIGH BYTE)
        RET                ;POP & JUMP
NEXT:   INC    DPTR
        INC    DPTR
        INC    DPTR
        JMP    SCAN

```

PAGE

```

HELP:   MOV    DPTR,#MHELP
        CALL   PDATA
        JMP    WARM

```


;*****

```

DUMP:  CALL    BADDR
      JNC     DMP3
      CLR     ACC.5           ;CONVERT LOWER TO UPPER CASE
      CJNE    A,#'I',DMP1
      JMP     DUMPI           ;DUMP INTERNAL RAM

DMP1:  JMP     WARM

DMP3:  MOV     FIRST,HIBYTE   ;DUMP PROGRAM MEMORY
      MOV     (FIRST+1),LOBYTE
      MOV     LAST,#0FFH
      MOV     (LAST+1),#0FFH
      CJNE    A,#CR,DMP4     ;DEFAULT IF NO END ADDRESS ENTERED
      JMP     (NEWLIN-6)

DMP4:  CALL    BADDR2
      JB      F0,DMP5         ;CARRIAGE RETURN READ
      MOV     LAST,HIBYTE
      MOV     (LAST+1),LOBYTE
DMP5:  MOV     DPTR,#MINDEX
      LCALL   PDATA

NEWLIN:
      MOV     DPH,FIRST
      MOV     DPL,(FIRST+1)
      ANL     DPL,#0F0H
      CALL    CRLF
      MOV     R0,DPH
      CALL    OUTR0           ;DISPLAY ADDRESS
      MOV     R0,DPL
      CALL    OUTR0
      CALL    OUT2S

SPACES: MOV     A,(FIRST+1)   ;BLANK LOCATIONS BEFORE FIRST
      XRL     A,DPL
      JZ      MORE           ;JUMP IF EQUAL
      CALL    OUT3S
      INC     DPTR
      CALL    MIDCHK
      JMP     SPACES

MORE:  CALL    OUTC2HS
      MOV     A,LAST
      CJNE    A,DPH,MOREC
      MOV     A,(LAST+1)
      CJNE    A,DPL,MOREC
      JMP     ENDHEX

MOREC: INC     DPTR
      CALL    MIDCHK
      JZ      DASCII         ;NOW DUMP ASCII
      JMP     MORE

ENDHEX: INC     DPTR
      CALL    MIDCHK
      JZ      DASCII         ;NOW DUMP ASCII (LAST LINE)
      CALL    OUT3S
      JMP     ENDHEX

DASCII: CALL    OUT4S         ;SHOW ASCII EQUIVALENT
      MOV     DPH,FIRST
      MOV     DPL,(FIRST+1)
      ANL     DPL,#0F0H
DASCI1: MOV     A,(FIRST+1)
      XRL     A,DPL
      JZ      DASCII2
      CALL    OUTS
      INC     DPTR
      CALL    MIDCHK
      JMP     DASCII1

DASCII2: CLR     A
      MOVC    A,@A+DPTR
      MOV     B,A
      CLR     C

```

(continued)

```

SUBB    A,#SPACE
JC      BAD3
SUBB    A,#(7FH-SPACE)
JC      OK3
BAD3:   MOV    B,#DOT
OK3:    MOV    A,B
        CALL   OUTCH

```

```

MOV     A, LAST
CJNE    A, DPH, DASC13
MOV     A, (LAST+1)
CJNE    A, DPL, DASC13
JMP     WARM

```

```

DASC13: INC    DPTR
        CALL   MIDCHK
        JNZ    DASC12
        MOV    A,#16
        ADD    A,(FIRST+1)
        ANL    A,#0F0H
        JNC    $+4
        INC    FIRST
        MOV    (FIRST+1),A
        JMP    NEWLIN

```

```

TABENT    EQU    11    ;LENGTH OF SFR TABLE ENTRY
OFF1      EQU    6     ;OFFSET TO HEX ADDRESS
OFF2      EQU    5     ;OFFSET TO FETCH CONTENTS
OFF3      EQU    8     ;OFFSET TO CHANGE CONTENTS

```

DUMPI:

```

CALL     BADDR
JNC      $+5
LJMP     WARM

```

```

MOV      FIRST, LOBYTE
MOV      LAST, #0FFH
XRL      A, #CR
JZ       (LINE-6)
CALL     BADDR2
JB       F0, $+6    ;CARRIAGE RETURN READ
MOV      LAST, LOBYTE
MOV      A, #7FH    ;TOP OF RAM
CLR      C

```

```

SUBB     A, FIRST
JC       DSFR      ;DONE RAM--DO SFR'S
MOV      DPTR, #MINDEX
LCALL    PDATA

```

```

LINE:    CALL    CRLF
        MOV     A, #7FH    ;TOP OF RAM
        CLR     C

```

```

SUBB     A, FIRST
JC       DSFR      ;DONE RAM--DO SFR'S
MOV      R0, FIRST
ANL      REG0, #0F0H ;8=R0

```

```

CALL     OUTR03S    ;DISPLAY ADDRESS
SPICES:  MOV     A, FIRST ;SPACES BEFORE BEGINNING

```

```

XRL      A, R0
JZ       NXTBYT
CALL     OUT3S
INC      R0
MOV      A, #07H
ANL      A, R0
JNZ      $+5
LCALL    OUTS        ;MIDDLE OF LINE
JMP      SPICES

```

```

NXTBYT: CALL    OUT2H
        CALL    OUTS
        MOV     A, R0
        XRL     A, LAST
        JNZ     $+5
        LJMP    WARM

```



```

INC      R0
MOV      A,#07H
ANL      A,R0
JNZ      $+5
LCALL    OUTS          ;MIDDLE OF LINE
MOV      A,#0FH
ANL      A,R0
JNZ      NXTBYT        ;NOT END OF LINE
MOV      A,FIRST
ANL      A,#0F0H
ADD      A,#10H
MOV      FIRST,A
JMP      LINE

DSFR:    MOV      DPTR,#(SFRTAB-TABENT)
MOV      R1,#6
CALL     CRLF
DSFR1:   MOV      A,#TABENT
ADD      A,DPL
MOV      DPL,A
JNC      $+4
INC      DPH
MOV      A,#OFF1        ;OFFSET TO SFR HEX LOCATION
MOVC     A,@A+DPTR
CLR      C
SUBB     A,FIRST
JC       DSFR1          ;NOT TO FIRST YET

DSFR5:   MOV      A,#OFF1
MOVC     A,@A+DPTR        ;GET HEX LOCATION OF SFR
MOV      R0,FIRST
XRL      A,R0
JZ       DSFR10
INC      FIRST          ;WASN'T A VALID SFR
MOV      A,FIRST
JZ       (LINE-6)        ;BACK TO RAM
CJNE     A,LAST,DSFR5
JMP      WARM           ;DONE

DSFR10:  CALL     OUTR0
MOV      A,#(' '=')
CALL     OUTCH
CALL     PDATA          ;OUTPUT LOCATION LABEL (ALTERS DPTR)
MOV      A,#(': ':')
CALL     OUTCH
MOV      A,#LOW(DSFR20)
PUSH     ACC
MOV      A,#HIGH(DSFR20)
PUSH     ACC
MOV      A,#(OFF2-4)    ;OFFSET TO "MOV R0,SFR"
JMP      @A+DPTR

DSFR20:  CALL     OUTR0
CALL     OUT2S
DJNZ     R1,DSFR30
MOV      R1,#6
CALL     CRLF
DSFR30:  INC      FIRST
MOV      A,#(TABENT-4)
ADD      A,DPL
MOV      DPL,A
JNC      $+4
INC      DPH
MOV      A,FIRST
JNZ      $+5
LJMP     LINE
DEC      A
XRL      A,LAST
JNZ      DSFR5
JMP      WARM

```

(continued)

```
COPY:  CALL  THRADR
        CALL  CPY
        JMP   WARM
```

```
VERIFY: CALL  THRADR
VER10:  MOV    DPH,FIRST
        MOV    DPL,(FIRST+1)
        CLR    A
        MOVC   A,@A+DPTR
        MOV    R1,A
        MOV    DPH,TO
        MOV    DPL,(TO+1)
        CLR    A
        MOVC   A,@A+DPTR
        MOV    R2,A
        XRL    A,R1
        JZ     VER20          ;JUMP IF EQUAL
        CALL   CRLF
        MOV    R0,#FIRST
        CALL   OUT2H
        MOV    R0,#(FIRST+1)
        CALL   OUT2H
        CALL   OUT2S
        MOV    R0,#REG1       ;R1
        CALL   OUT2H
        CALL   OUT4S
        MOV    R0,#TO
        CALL   OUT2H
        MOV    R0,#(TO+1)
        CALL   OUT2H
        CALL   OUT2S
        MOV    R0,#REG2       ;R2
        CALL   OUT2H
VER20:  MOV    A,FIRST
        CJNE   A,LAST,VER30
        MOV    A,(FIRST+1)
        CJNE   A,(LAST+1),VER30
        JMP    WARM
VER30:  MOV    R0,#(FIRST+1)
        CALL   INC16
        MOV    R0,#(TO+1)
        CALL   INC16
        JMP    VER10
```

```
ALTER:  CALL   BADDR
        JNC    ALT05
        JNB    F0,ALTEND      ;ERROR
        CLR    ACC.5          ;CONVERT LOWER TO UPPER CASE
        CJNE   A,#('I'),ALTEND ;ERROR
        JMP    SUBSTUT

ALT05:  MOV    FIRST,HIBYTE
        MOV    (FIRST+1),LOBYTE
ALT10:  CALL   CRLF
        MOV    R0,#FIRST      ;SHOW ADDRESS
        CALL   OUT2H
        INC    R0
        CALL   OUT2H
ALT15:  CALL   OUTS
        MOV    DPH,FIRST
        MOV    DPL,(FIRST+1)
        CLR    A
        MOVC   A,@A+DPTR      ;GET DATA
        MOV    R1,A
        MOV    R0,#REG1       ;SHOW DATA
        CALL   OUT2H
        MOV    A,#('-')
        CALL   OUTCH
        CALL   BYTES
        JNC    ALT20
        XRL    A,#BACKSP
```



```

        JZ      BACKUP
        XRL     A, #BACKSP      ;RESTORE A
        XRL     A, #DOT
        JZ      BACKUP
ALTEND:  JMP     WARM           ;ERROR

ALT20:   JB      F0, ALT30      ; <CR> OR SPACE
        MOV     A, LOBYTE
        MOVX    @DPTR, A       ;CHANGE DATA
ALT30:   MOV     R0, #(FIRST+1)
        CALL    INC16
        MOV     A, B
        XRL     A, #CR
        JZ      ALT10
        MOV     A, (FIRST+1)
        ANL     A, #07H
        JZ      ALT10
        JMP     ALT15

BACKUP:  MOV     R0, #(FIRST+1)
        CALL    DEC16
        JMP     ALT10

;*****

MODIFY   CALL    BADDR
        MOV     DPH, HIBYTE
        MOV     DPL, LOBYTE
MOD10:   CALL    INCH
        CJNE    A, #BACKSP, NOTBS
        DEC     DPL
        MOV     A, #0FFH
        CJNE    A, DPL, MOD10
        DEC     DPH
        JMP     MOD10

NOTBS:   CJNE    A, #EOT, $+6
        LJMP    WARM
        MOVX    @DPTR, A
        INC     DPTR
        JMP     MOD10

;*****

SUBSTUT:
        CALL    BADDR
        MOV     R0, LOBYTE
IF010:   MOV     A, #7FH        ;IF ADDRESS >= 80H OR <= F0H
        CLR     C
        SUBB    A, R0
        JNC     ELS010
        MOV     A, #0F0H
        CLR     C
        SUBB    A, R0
        JC      ELS010
THN010:  MOV     DPTR, #(SFRTAB-TABENT)
SUB60:   MOV     A, #TABENT      ;INCREMENT TABLE POINTER
        ADD     A, DPL
        MOV     DPL, A
        JNC     $+4
        INC     DPH
        MOV     A, #OFF1        ;FIND 1ST ENTRY >= R0
        MOVC    A, @A+DPTR
        CLR     C
        SUBB    A, R0
        JC      SUB60          ;NOT FOUND YET
SUB65:   MOV     A, #OFF1        ;INCREMENT R0 TO LINE IN TABLE
        MOVC    A, @A+DPTR
        XRL     A, R0
        JZ      SUB70          ;FOUND IT
        INC     R0
        JMP     SUB65

ELS010:  JMP     LS010          ;RELAY STATION

```

(continued)

```

SUB70:  CALL    CRLF                ;DISPLAY ADDRESS
        CALL    OUTR03S
        CALL    PDATA              ;ALTERS DPTR (ADDS 4)
        MOV     A,#(':' )
        CALL    OUTCH
        MOV     REG1,R0            ;MOV     R1,R0
        MOV     A,#LOW(SUB80)
        PUSH    ACC
        MOV     A,#HIGH(SUB80)
        PUSH    ACC
        MOV     A,#(OFF2-4)
        JMP     @A+DPTR            ;CRASHES R0

SUB80:  CALL    OUTR0                ;DISPLAY DATA
        MOV     A,#('-' )
        CALL    OUTCH
        MOV     R0,REG1            ;RESTORE R0
        CALL    BYTES
IF020:  JNC     ELS020              ;IF CHARACTER = DOT OR BACKSPACE
        MOV     A,B
        XRL     A,#BACKSP
        JZ      THN020
        MOV     A,B
        XRL     A,#DOT
        JZ      THN020
        JMP     WARM                ;ERROR

THN020:                                ;THEN BACKUP
IF030:  CJNE    R0,#80H,ELS030
THN030:  MOV     R0,#7FH
        JMP     IF010

ELS030:  MOV     A,DPL
        CLR     C
        SUBB    A,#(TABENT+4)
        MOV     DPL,A
        JNC     $+4
        DEC     DPH
        MOV     A,#(OFF1)
        MOVC    A,@A+DPTR
        MOV     R0,A
        JMP     IF010

ELS020:  JB      F0,SUB90
        SETB    F0
        MOV     R0,LOBYTE
        MOV     A,#LOW(SUB90)
        PUSH    ACC
        MOV     A,#HIGH(SUB90)
        PUSH    ACC
        MOV     A,#(OFF3-4)
        JMP     @A+DPTR            ;CRASHES R0

SUB90:  MOV     R0,REG1            ;RESTORE R0
IF035:  JB      F0,NDI035
THN035:  MOV     DPTR,#MNONO
        CALL    PDATA

NDI035:
IF040:  CJNE    R0,#0F0H,NDI040    ;IF R0 = 0F
THN040:  MOV     R0,#0FFH          ;THEN R0 = FFH
NDI040:  INC     R0
        JMP     IF010

LS010:  CALL    CRLF
        CALL    OUTR03S
SUB20:  CALL    OUT2H
        MOV     A,#('-' )
        CALL    OUTCH
        MOV     REG1,R0
        CALL    BYTES              ;clobbers r0
        MOV     R0,REG1
IF050:  JNC     ELS050              ;IF CHAR. = BACKSP. OR DOT
        MOV     A,B
        XRL     A,#BACKSP
        JZ      THN050

```



```

MOV      A,B
XRL      A,#DOT
JZ       THN050
JMP      WARM

THN050:                                ; THEN BACKUP
IF060:   MOV      A,R0
        JNZ      NDI060                ; IF R0 = 0
THN060:   MOV      R0,#0F1H            ; THEN R0 = F1H
NDI060:
NDI050:
F010:    DEC      R0
        JMP      IF010                ; ALSO USED FOR RELAY

ELS050:
IF070:   JB       F0,NDI070            ; IF NOT SPACE OR CR
THN070:   MOV      A,LOBYTE            ; THEN CHANGE BYTE
        MOV      @R0,A
NDI070:   INC      R0
        MOV      A,#07H
        ANL      A,R0
        JZ       F010
        CALL     OUT2S
        JMP      SUB20

;*****

INSERT:  CALL     THRADR
        MOV      DPH,FIRST
        MOV      DPL,(FIRST+1)
INS10:   MOV      A,(TO+1)
        MOVX     @DPTR,A
        MOV      A,DPH
        CJNE     A,LAST,INS20
        MOV      A,DPL
        CJNE     A,(LAST+1),INS20
        JMP      WARM

INS20:   INC      DPTR
        JMP      INS10

;*****

BREAK:   ; REMOVE OLD BREAKPOINT
        MOV      DPTR,#BYTENUM
        CLR      A
        MOVC     A,@A+DPTR
        MOV      R1,A                ; SAVE OLD BYTENUM IN R1
IF100:   JZ       NDI100                ; IF THERE IS AN OLD BREAKPOINT
THN100:   MOV      FIRST,#HIGH(BYTENUM+1) ; THEN REMOVE IT
        MOV      FIRST+1,#LOW(BYTENUM+1)
        MOV      LAST,#HIGH(BYTENUM)
        MOV      A,#LOW(BYTENUM)
        ADD      A,R1                ; ADD OLD BYTENUM
        MOV      LAST+1,A
        JNC      $+4
        INC      LAST
        MOV      DPTR,#(RAMEND-1)
        CLR      A
        MOVC     A,@A+DPTR
        MOV      TO,A
        MOV      DPTR,#RAMEND
        CLR      A
        MOVC     A,@A+DPTR
        CLR      C
        SUBB     A,R1                ; SUBTRACT OLD BYTENUM
        MOV      TO+1,A
        JNC      $+4
        DEC      TO
        CALL     CPY
        CLR      A                ; CLEAR END OF RAM
        MOV      DPTR,#BYTENUM
        MOV      R0,#(RAMEND-BYTENUM+1-3) ; LEAVE JUMP INSTRUCTION
CLEAR1:  MOVX     @DPTR,A
        INC      DPTR
        DJNZ     R0,CLEAR1

```

(continued)

```

NDI100: CALL    BYTES          ;INSTALL NEW BREAKPOINT
IF150:  JNC     NDI150        ;IF NOT VALID HEX
        JMP     WARM          ;THEN END

NDI150:
IF155:  JNB     F0,NDI155     ;IF CR
        XRL     A,#SPACE
        JZ      NDI100       ;AND NOT SPACE
        JMP     WARM          ;THEN END

NDI155: MOV     FIRST,HIBYTE
        MOV     FIRST+1,LOBYTE
        MOV     DPTR,#BYTENUM
        MOV     A,#3          ;IN CASE OF DEFAULT
        MOVX    @DPTR,A
        MOV     R1,A          ;SAVE DEFAULT NEW BYTENUM
IF160:  MOV     A,B
        XRL     A,#CR
        JZ      NDI160
THN160: CALL    INHEX         ;get BYTENUM
IF170:  JNC     ELS170        ;if not valid hex
THN170: MOV     A,B          ;then check for space or CR
        XRL     A,#SPACE
        JZ      THN160
        MOV     A,B
        XRL     A,#CR
        JZ      NDI170
        MOV     DPTR,#BYTENUM
        CLR     A
        MOVX    @DPTR,A
        JMP     WARM

ELS170: MOV     DPTR,#BYTENUM ;else accept only 3, 4 or 5
        MOVX    @DPTR,A
        MOV     R1,A          ;SAVE NEW BYTENUM
        MOV     A,#3
        JC      BAD4
        SUBB    A,#(6-3)
        JNC     BAD4
        JMP     NDI170

BAD4:   MOV     DPTR,#BYTENUM
        CLR     A
        MOVX    @DPTR,A
        JMP     WARM

NDI170:
NDI160:                                     ;FIRST,FIRST+1 AND HIBYTE,LOBYTE
                                           ; BOTH CONTAIN BREAK ADDRESS
                                           ;R1 CONTAINS NEW BYTENUM
        MOV     DPTR,#(RAMEND-2)
        MOV     A,#02H        ;"LJMP"
        MOVX    @DPTR,A
        MOV     A,FIRST+1
        ADD     A,R1          ;ADD BYTENUM
        MOV     LAST+1,A
        MOV     DPTR,#RAMEND
        MOVX    @DPTR,A
        MOV     A,FIRST
        JNC     $+3
        INC     A
        MOV     LAST,A
        MOV     DPTR,#(RAMEND-1)
        MOVX    @DPTR,A
        MOV     R0,#(LAST+1)
        CALL    DEC16         ;ADJUST (LAST, LAST+1)
        MOV     TO,#HIGH(BYTENUM+1)
        MOV     TO+1,#LOW(BYTENUM+1)
        CALL    CPY          ;SAVE INSTRUCTIONS IN END OF RAM
        MOV     DPL,LOBYTE
        MOV     DPH,HIBYTE   ;INSERT JUMP TO BRKPT
        MOV     A,#02H        ;"LJMP"
        MOVX    @DPTR,A
        INC     DPTR

```



```

MOV    A,#HIGH(BRKPT)
MOVX   @DPTR,A
INC    DPTR
MOV    A,#LOW(BRKPT)
MOVX   @DPTR,A
JMP    WARM

```

```

;*****

```

```

BRKPT:  PUSH    ACC
        PUSH    PSW
        PUSH    B
        PUSH    DPH
        PUSH    DPL
        MOV     STACK,SP          ;save current stack level for cool start
        SETB    RS0              ;SELECT REGISTER BANK 1
        CLR     RS1
        MOV     DPTR,#MBRK1
        CALL    PDATA
        MOV     DPTR,#(BYTENUM)
        CLR     A
        MOVC    A,@A+DPTR
        MOV     R0,A
        MOV     DPTR,#(RAMEND)
        CLR     A
        MOVC    A,@A+DPTR
        CLR     C
        SUBB    A,R0
        MOV     B,A
        MOV     DPTR,#(RAMEND-1)
        CLR     A
        MOVC    A,@A+DPTR
        MOV     R0,A
        JNC     $+3
        DEC     R0
        CALL    OUTR0
        MOV     R0,B
        CALL    OUTR0
        MOV     DPTR,#MBRK2
        CALL    PDATA
        MOV     A,STACK
        CLR     C
        SUBB    A,#4
        MOV     R0,A
        CALL    OUT2H
        MOV     DPTR,#MBRK3
        CALL    PDATA
        INC     R0
        CALL    OUT2H
        MOV     DPTR,#MBRK4
        CALL    PDATA
        INC     R0
        CALL    OUT2H
        MOV     DPTR,#MBRK5
        CALL    PDATA
        INC     R0
        CALL    OUT2H
        INC     R0
        CALL    OUT2H
        MOV     DPTR,#MBRK6
        CALL    PDATA
        MOV     R0,#STACK
        CALL    OUT2H
        JMP     WARM

```

```

;*****

```

```

GO:     CALL    BYTES
        JNC     $+5
        LJMP    WARM

        JNB     F0,GXXXX
        CJNE    A,#CR,GO          ;LOOK AGAIN IF SPACE
        MOV     HIBYTE,#HIGH(BYTENUM+1)
        MOV     LOBYTE,#LOW(BYTENUM+1)
        MOV     SP,STACK          ;restore stack if necessary

```

(continued)

```

      POP      DPL
      POP      DPH
      POP      B
      POP      PSW
      POP      ACC

GXXXX:
      PUSH     LOBYTE
      PUSH     HIBYTE
      RET

```

```

;*****

```

```

HEXMATH:

```

```

      CALL     BADDR
      JNC      $+5
      LJMP     WARM

      MOV      FIRST,HIBYTE
      MOV      (FIRST+1),LOBYTE
      CALL     BADDR
      JNC      $+5
      LJMP     WARM

      MOV      A,(FIRST+1)
      ADD      A,LOBYTE
      MOV      (LAST+1),A      ;STORE LOW BYTE OF SUM
      MOV      A,FIRST
      ADDC     A,HIBYTE
      MOV      LAST,A          ;STORE HIGH BYTE OF SUM

      MOV      A,(FIRST+1)
      CLR      C
      SUBB     A,LOBYTE
      MOV      (TO+1),A        ;STORE LOW BYTE OF DIFFERENCE
      MOV      A,FIRST
      SUBB     A,HIBYTE
      MOV      TO,A            ;STORE HIGH BYTE OF DIFFERENCE
      CALL     CRLF
      MOV      R0,#FIRST      ;POINT TO ADDEND
      CALL     OUT4HS
      MOV      A,'#+'
      CALL     OUTCH
      CALL     OUTS
      MOV      R0,#HIBYTE     ;POINT TO AUGEND
      CALL     OUT4HS
      MOV      A,'#='
      CALL     OUTCH
      CALL     OUTS
      MOV      R0,#LAST       ;POINT TO SUM
      CALL     OUT4HS
      CALL     CRLF
      MOV      R0,#FIRST      ;POINT TO SUBTRAHEND
      CALL     OUT4HS
      MOV      A,'#-'
      CALL     OUTCH
      CALL     OUTS
      MOV      R0,#HIBYTE     ;POINT TO MINUEND
      CALL     OUT4HS
      MOV      A,'#='
      CALL     OUTCH
      CALL     OUTS
      MOV      R0,#TO         ;POINT TO DIFFERENCE
      CALL     OUT4HS
      JMP      WARM

```

```

;*****

```

```

JUMPTABL:

```

```

      MOV      DPTR,#MJUMP
      CALL     PDATA
      JMP      WARM

```

```

PAGE

```

```

;*****

```

```

;ROUTINE      PDATA
;              WRITES A MESSAGE TO THE TERMINAL

```



```
; ENTER WITH DPTR POINTING TO BEGINNING OF MESSAGE
; AND 04H AT END OF MESSAGE.
```

```
;
; PDATA1: CALL    OUTCH
;          INC     DPTR
; PDATA:  CLR     A
;          MOVC    A,@A+DPTR
;          CJNE    A,#EOT,PDATA1
;          RET
```

```
;*****
```

```
;ROUTINE      IN
;          READ AN 8-BIT CHAR FROM 8251A UART
;
; IN:         PUSH    DPH
;             PUSH    DPL
;             MOV     DPTR,#UARTCONT
;             MOVX    A,@DPTR
;             JNB     ACC.1,$-1      ;WAIT FOR RX RDY
;             DEC     DPL            ;POINT TO DATA REGISTER
;             MOVX    A,@DPTR
;             POP     DPL
;             POP     DPH
;             RET
```

```
;*****
```

```
;ROUTINE      INCH
;          READ A CHARACTER, ZERO HIGH BIT, & ECHO BACK TO TERMINAL
;
; INCH:       CALL    IN              ;READ 8-BIT CHAR.
;             ANL     A,#7FH          ;ZERO HIGH BIT
;             JMP     OUT             ;ECHO BACK
```

```
;*****
```

```
;ROUTINE      OUTCH
;          OUTPUT A CHARACTER
;
; OUTCH:      CALL    OUT
;             JNC     NOINPU          ;TEST FOR INPUT DURING OUTPUT
;             CALL    IN              ;GET RID OF THE CHARACTER
;             JMP     COOL            ;RESET STACK POINTER
```

```
NOINPU: RET
```

```
;*****
```

```
;ROUTINE      OUT
;          SEND "A" REGISTER TO UART
;          SET CARRY IF RX BUFFER IS FULL
;
; OUT:        PUSH    DPH
;             PUSH    DPL
;             PUSH    ACC             ;SAVE OUTPUT BYTE
;             MOV     DPTR,#UARTCONT
;             MOVX    A,@DPTR
;             JNB     ACC.0,$-1      ;WAIT FOR TX EMPTY
;             MOV     C,ACC.1         ;MOV RX RDY TO CARRY
;             DEC     DPL            ;POINT TO DATA REGISTER
;             POP     ACC            ;RESTORE OUTPUT BYTE
;             MOVX    @DPTR,A        ;AND OUTPUT IT
;             POP     DPL
;             POP     DPH
;             RET
```

```
PAGE      30
```

```
;*****
```

```
;ROUTINE      INHEX
;          READS 1 ASCII HEX CHAR, CONVERTS TO BINARY IN ACC. &
;          STORES ORIGINAL CHAR IN B REG.
;          CARRY SET IF NOT VALID HEX, CLEARED OTHERWISE.
;
;          ;
```

(continued)

```

INHEX:  CALL    INCH          ;GET CHARACTER
        MOV     B,A           ;STORE IN B
        CJNE    A,#CR,NOTCR   ;IF <CR> SEND <LF> ALSO
        MOV     A,#LF
        CALL    OUTCH
NOTCR:   MOV     A,B           ;RESTORE CHARACTER
        JNB     ACC.6,$+5      ;IF LOWER CASE
        CLR     ACC.5         ;CONVERT TO UPPER CASE
        CLR     C
        SUBB    A,#'0'
        JC      BAD           ; ACC < '0'
        SUBB    A,#10
        JNC     $+7           ; ACC > '9'
        ADD     A,#10         ;RESTORE
        LJMP    GOOD          ; '0' <= ACC <= '9'

        ADD     A,#('0'+10-'A'+10) ;CORRECT FOR (-'0'-10) & MAP 'A' INTO 10
        JB      ACC.7,BAD      ; '9' < ACC < 'A'
        CLR     C
        SUBB    A,#16
        JNC     BAD           ; ACC > 'F'
        ADD     A,#16
GOOD:    CLR     C
        RET
BAD:     SETB    C
        RET

```

```

;*****
;
;ROUTINE      BYTES
;  READ IN TWO BYTE HEX NUMBER TO HIBYTE,LOBYTE
;  LAST CHARACTER READ RETURNED IN B REGISTER
;
;  ERROR CODES:          CARRY:      F0:
;  LEADING <CR> OR SPACE      0          1      (CHAR IN ACC)
;  LEADING NONHEX CHARACTER   1          1      (CHAR IN ACC)
;  OTHER NONHEX CHARACTER     1          0      (CHAR IN ACC)
;

```

```

BYTES:  SETB    F0            ;NO HEX READ YET
        CLR     A
        MOV     HIBYTE,A
        MOV     LOBYTE,A
MORE1:  CALL    INHEX
        JNC     OK1          ;JUMP IF HEX DIGIT
        MOV     A,B          ;LOOK AT ASCII
        CJNE    A,#CR,$+6
        LJMP    CR1
CR1:    CJNE    A,#SPACE,BAD1
        CLR     C
        RET
OK1:    CLR     F0
        MOV     R0,#LOBYTE   ;POINTER
        XCHD    A,@R0        ;PUT 0 DIGIT IN LOW END OF LOBYTE
        SWAP    A            ;ACC NOW HAS DIGIT 1 IN HIGH END
        XCHD    A,@R0        ;ACC NOW HAS DIGITS 1,0
        XCH     A,@R0        ;ACC HAS DIGITS 2,X;LOBYTE DONE.
        DEC     R0           ;POINT TO HIBYTE
        XCHD    A,@R0        ;ACC NOW HAS DIGITS 2,3
        SWAP    A            ;ACC HAS DIGITS 3,2
        XCH     A,@R0        ;HIBYTE DONE
        JMP     MORE1        ;LOOK FOR ANOTHER DIGIT
BAD1:   SETB    C
        RET

```

PAGE 19

```

;*****
;
;ROUTINE      BUILD ADDRESS
;  READ IN A 16-BIT HEX NUMBER TO HIBYTE,LOBYTE.
;  RETURNS WITH CARRY & F0 SET IF A SECOND COMMAND CHARACTER
;  IS FOUND. CHARACTER WILL BE IN ACCUMULATOR.
;  RETURNS WITH CARRY SET AND F0 CLEAR IF NON-HEX.
;

```



```

BADDR: CALL    BYTES
      JNC     ADDR0K
      JB      F0,ADDRNOK      ;SECOND COMMAND CHARACTER
      JMP     WARM            ;ERROR
ADDR0K: JB      F0,BADDR      ;LEADING SPACE OR CR
ADDRNOK:
      RET

```

```

;*****
;

```

```

;ROUTINE      BUILD ADDRESS #2
;
;  READ IN A 16-BIT HEX NUMBER TO HIBYTE,LOBYTE.
;  IGNORES LEADING SPACES.  RETURNS WITH F0 SET IF
;  A LEADING CARRIAGE RETURN IS FOUND.
;

```

```

BADDR2: CALL    BYTES
      JNC     $+5
      LJMP    WARM            ;NON-HEX

      JNB     F0,ADDR0K2      ;NUMBER READ
      CJNE    A,#CR,BADDR2    ;IGNORE LEADING SPACE
ADDR0K2:
      RET

```

```

;*****
;

```

```

;ROUTINE      OUTPUT SPACES
;
;  OUTPUTS 1, 2, 3, OR 4 SPACES
;

```

```

OUT4S: CALL    OUTS
OUT3S: CALL    OUTS
OUT2S: CALL    OUTS
OUTS:  MOV     A,#SPACE
      JMP     OUTCH

```

```

;*****
;

```

```

;ROUTINE      OUTPUT CODE, TWO HEX, SPACE
;
;  OUTPUTS PROGRAM MEMORY BYTE POINTED OUT BY DPTR
;  AS TWO ASCII CHARACTERS FOLLOWED BY A SPACE.
;  USES B REGISTER AS TEMP. STORE
;

```

```

OUTC2HS:
      CLR     A
      MOVC    A,@A+DPTR
      MOV     B,A            ;TEMP STORE
      CALL    OUTHL
      MOV     A,B
      CALL    OUTHR
      JMP     OUTS

```

```

;*****
;

```

```

;ROUTINE      MIDCHK
;
;  INSERTS A SPACE IF LOW NYBBLE OF DPTR = 8.
;  RETURNS WITH LOW NYBBLE IN ACC.
;  DESTROYS B REGISTER.
;

```

```

MIDCHK: MOV     A,DPL
      ANL     A,#0FH
      XRL     A,#08H
      JNZ     NOTMID
      XCH     A,B
      CALL    OUTS
      XCH     A,B
NOTMID: XRL     A,#08H      ;RESTORE A
      RET

```

```

      PAGE    14

```

```

;*****
;

```

```

;ROUTINES      OUTPUT HEX LEFT
;
;  OUTPUT HEX RIGHT
;  CONVERTS A NYBBLE IN ACC. TO ASCII AND SENDS IT
;

```

(continued)

July

```
OUTH:  SWAP      A
OUTHR:  ANL       A,#0FH
        JNB      ACC.3,H2      ;<8
        JB       ACC.2,H1      ;>=C
        JNB      ACC.1,H2      ;<A
H1:     ADD       A,#07H
H2:     ADD       A,#('0')      ;CONVERT TO ASCII
        JMP      OUTCH
```

PAGE 9

```
;
;ROUTINE      OUTPUT TWO HEX
;      OUTPUTS HEX CONTENTS OF LOCATION POINTED OUT BY R0
;
OUT2H:  MOV      A,@R0
        CALL     OUTHL
        MOV      A,@R0
        JMP      OUTHR
```

```
;
;ROUTINE      CRLF
;      SENDS A CARRIAGE RETURN AND A LINE FEED
;
CRLF:   MOV      A,#CR
        CALL     OUTCH
        MOV      A,#LF
        JMP      OUTCH
```

```
;
;ROUTINE      DECREMENT 16
;      DECREASES A 16-BIT NUMBER POINTED OUT BY R0.
;      ENTER WITH LOW BYTE @R0 & HIGH BYTE @(R0-1).
;      CARRY SET ON OVERFLOW, CLEARED OTHERWISE.
;
DEC16:  CLR      C
        DEC      @R0
        MOV      A,@R0
        CPL      A
        JNZ      DECEND
        DEC      R0
        DEC      @R0
        MOV      A,@R0
        CPL      A
        JNZ      DECEND
        SETB     C
DECEND:  RET
```

PAGE 17

```
;
;ROUTINE      INCREMENT 16
;      INCREASES A 16-BIT NUMBER POINTED OUT BY R0.
;      ENTER WITH LOW BYTE @R0 & HIGH BYTE @(R0-1).
;      CARRY SET ON OVERFLOW, CLEARED OTHERWISE.
;
INC16:  CLR      C
        INC      @R0
        MOV      A,@R0
        JNZ      INCEND
        DEC      R0
        INC      @R0
        MOV      A,@R0
        JNZ      INCEND
        SETB     C
INCEND:  RET
```

PAGE 20

```
;
;ROUTINE      THREE ADDRESSES
;      GETS THREE 16-BIT HEX NUMBERS AND STORES THEM IN
;      "FIRST", "LAST", AND "TO" RESPECTIVELY.
```



```
;
THRADR: CALL    BADDR
        JC      THRERR
        MOV     FIRST,HIBYTE
        MOV     (FIRST+1),LOBYTE
        CALL    BADDR
        JC      THRERR
        MOV     LAST,HIBYTE
        MOV     (LAST+1),LOBYTE
        CALL    BADDR
        JC      THRERR
        MOV     TO,HIBYTE
        MOV     (TO+1),LOBYTE
        RET
THRERR: JMP     COOL
```

```
*****
```

```
;ROUTINE      COPY
;              COPIES PROGRAM MEMORY LOCATED "FIRST" TO "LAST"
;              TO RAM LOCATION STARTING AT "TO"
```

```
;
CPY:  MOV     DPH,FIRST
      MOV     DPL,(FIRST+1)
      CLR     A
      MOVC    A,@A+DPTR
      MOV     DPH,TO
      MOV     DPL,(TO+1)
      MOVX    @DPTR,A          ;PUT DATA
      MOV     A,FIRST
      CJNE    A,LAST,CPY2
      MOV     A,(FIRST+1)
      CJNE    A,(LAST+1),CPY2
      RET                      ;DONE
```

```
CPY2:  MOV     R0,#(FIRST+1)
      CALL    INC16
      MOV     R0,#(TO+1)
      CALL    INC16
      JMP     CPY
```

```
*****
```

```
;ROUTINE      OUT, 2 HEX, 3 SPACES
;              OUTPUTS LOCATION POINTED OUT BY R0 FOLLOWED BY 3 SPACES
```

```
;
OUT2H3S:
      CALL    OUT2H
      JMP     OUT3S
```

```
PAGE      11
```

```
*****
```

```
;ROUTINE      OUTPUT R0
;              OUTPUTS THE CONTENTS OF R0
;              (8051 CAN'T ACCESS SFR'S INDIRECTLY--UGH)
```

```
;
OUTR0:  MOV     A,R0
      CALL    OUTHL
      MOV     A,R0
      JMP     OUTHR
```

```
PAGE      9
```

```
*****
```

```
;ROUTINE      OUTPUT R0, 3 SPACES
;              OUTPUTS THE CONTENTS OF R0 AND THREE SPACES
```

```
;
OUTR03S:
      CALL    OUTR0
      JMP     OUT3S
```

```
*****
```

```
;ROUTINE      OUTPUT FOUR HEX, SPACE
```

(continued)

```

;      OUTPUTS TWO CONSECUTIVE INTERNAL MEMORY LOCATIONS,
;      THE LOWER OF WHICH (HIGH BYTE OF NUMBER) IS POINTED
;      OUT BY R0.
;
OUT4HS: CALL    OUT2H
        INC     R0
        CALL    OUT2H
        JMP     OUTS

```

PAGE

FUNTAB: ;FUNCTION TABLE

```

DB      'A'
DW      ALTER
DB      'B'
DW      BREAK
DB      'C'
DW      COPY
DB      'D'
DW      DUMP
DB      'G'
DW      GO
DB      'H'
DW      HELP
DB      'I'
DW      INSERT
DB      'J'
DW      JUMPTABL
DB      'M'
DW      MODIFY
DB      'V'
DW      VERIFY
DB      '#'
DW      HEXMATH
DB      00H ;END OF TABLE

```

SFRTAB: ;SPECIAL FUNCTION REGISTER TABLE

```

DB      'P0 ',04H
MOV     R0,P0
RET
CLR     F0
RET
DB      'SP ',04H
MOV     R0,SP
RET
CLR     F0
RET
DB      'DPL ',04H
MOV     R0,DPL
RET
MOV     DPL,R0
RET
DB      'DPH ',04H
MOV     R0,DPH
RET
MOV     DPL,R0
RET
DB      'PCON',04H
MOV     R0,PCON
RET
MOV     PCON,R0
RET
DB      'TCON',04H
MOV     R0,TCON
RET
MOV     TCON,R0
RET
DB      'TMOD',04H
MOV     R0,TMOD
RET
MOV     TMOD,R0
RET
DB      'TL0 ',04H

```



```

MOV    R0,TL0
RET
MOV    TL0,R0
RET
DB     'TL1 ',04H
MOV    R0,TL1
RET
MOV    TL1,R0
RET
DB     'TH0 ',04H
MOV    R0,TH0
RET
MOV    TH0,R0
RET
DB     'TH1 ',04H
MOV    R0,TH1
RET
MOV    TH1,R0
RET
DB     'P1 ',04H
MOV    R0,P1
RET
MOV    P1,R0
RET
DB     'SCON',04H
MOV    R0,SCON
RET
MOV    SCON,R0
RET
DB     'SBUF',04H
MOV    R0,SBUF
RET
MOV    SBUF,R0
RET
DB     'P2 ',04H
MOV    R0,P2
RET
CLR    F0
RET
DB     'IE ',04H
MOV    R0,IE
RET
MOV    IE,R0
RET
DB     'P3 ',04H
MOV    R0,P3
RET
MOV    P3,R0
RET
DB     'IP ',04H
MOV    R0,IP
RET
MOV    IP,R0
RET
DB     'PSW ',04H
MOV    R0,PSW
RET
MOV    PSW,R0
RET
DB     'ACC ',04H
MOV    R0,ACC
RET
MOV    ACC,R0
RET
DB     'B ',04H
MOV    R0,B
RET
MOV    B,R0
RET

```

```

;*****

```

```

MPROMPT: DB     CR,LF,'UGH:',04H

```

(continued)

```

MHHELP: DB      CR,LF,'REGISTER BANK 1 IS RESERVED'
DB      CR,LF,'LAST 9 BYTES OF EXT. RAM USED BY BREAK ROUTINE'
DB      CR,LF,'RAM @ 48H-50H IS USED BY MONITOR'
DB      CR,LF,'RAM ABOVE 50H IS USED FOR STACK',LF
DB      CR,LF,'COMMANDS ARE',LF
DB      CR,LF,'A SSSS',TAB,TAB,TAB,'ALTER EXTERNAL MEMORY'
DB      CR,LF,'AI SS',TAB,TAB,TAB,'ALTER INTERNAL MEMORY'
DB      CR,LF,'B {AAAA {#}}',TAB,TAB,'BREAK @ AAAA (#=3,4,OR 5)'
DB      CR,LF,'C SSSS FFFF TTTT',TAB,'COPY BLOCK OF MEMORY'
DB      CR,LF,'D SSSS {FFFF}',TAB,TAB,'DUMP PROGRAM MEMORY'
DB      CR,LF,'DI SS {FF}',TAB,TAB,'DUMP INTERNAL MEMORY'
DB      CR,LF,'G {AAAA}',TAB,TAB,'GO @ AAAA OR BREAKPOINT'
DB      CR,LF,'H',TAB,TAB,TAB,'HELP'
DB      CR,LF,'I SSSS FFFF HH',TAB,TAB,'INSERT "HH" INTO MEMORY'
DB      CR,LF,'J',TAB,TAB,TAB,'LIST JUMP TABLE'
DB      CR,LF,'M SSSS',TAB,TAB,TAB,'MODIFY EXTERNAL MEMORY (ASCII)'
DB      CR,LF,'V SSSS FFFF TTTT',TAB,'VERIFY MEMORY'
DB      CR,LF,'# MMMM NNNN ',TAB,TAB,'HEX ADDITION & SUBTRACTION'
DB      CR,LF,04H

MBRK1: DB      CR,LF,'BREAK AT LOCATION ',04H
MBRK2: DB      CR,LF,'ACC = ',04H
MBRK3: DB      '    PSW = ',04H
MBRK4: DB      '    B = ',04H
MBRK5: DB      '    DPTR = ',04H
MBRK6: DB      '    SP = ',04H

MINDEX: DB      CR,LF,'    0 1 2 3 4 5 6 7 8 9 A B C D E F',04H

MNONO: DB      CR,LF,'OOPS, THAT',27H,'S A NO-NO !',04H

MJUMP:
DB      CR,LF,'0026    COOL',TAB,'COOL START (RESET STACK POINTER)'
DB      CR,LF,'0029    WARM',TAB,'WARM START'
DB      CR,LF,'002C    IN',TAB,'GET A BYTE FROM THE UART'
DB      CR,LF,'002F    INCH',TAB,'GET A CHARACTER (ZERO PARITY)'
DB      CR,LF,'0032    INHEX',TAB,'GET A HEX CHAR (CARRY=NONHEX)'
DB      CR,LF,'0035    BYTES',TAB,'2 BYTE HEX TO HI/LOBYTE'
DB      CR,LF,'0038    BADDR',TAB,'BUILD ADDRESS'
DB      CR,LF,'003B    THRADR',TAB,'THREE ADDRESSES'
DB      CR,LF,'003E    OUT',TAB,'OUTPUT BYTE IN ACC.'
DB      CR,LF,'0041    OUTCH',TAB,'OUTPUT CHARACTER IN ACC.'
DB      CR,LF,'0044    OUTS',TAB,'OUTPUT SPACE'
DB      CR,LF,'0047    OUT2S',TAB,'OUTPUT 2 SPACES'
DB      CR,LF,'004A    CRLF',TAB,'OUTPUT [CR] AND [LF]'
DB      CR,LF,'004D    OUT2H',TAB,'OUTPUT LOC. POINTED BY R0'
DB      CR,LF,'0050    OUTR0',TAB,'OUTPUT CONTENTS OF R0'
DB      CR,LF,'0053    OUTC2HS',TAB,'OUTPUT PROG. MEM. POINTED'
DB      '    BY DPTR, AND SPACE'
DB      CR,LF,'0056    PDATA',TAB,'OUTPUT MESSAGE POINTED BY DPTR'
DB      CR,LF,'0059    INC16',TAB,'INCREMENT 2 BYTE NO.'
DB      '    (R0=LOW BYTE)'
DB      CR,LF,'005C    DEC16',TAB,'DECREMENT 2 BYTE NO.'
DB      '    (R0+1=HIGH BYTE)'
DB      CR,LF,'005F    CPY',TAB,'BLOCK COPY'
DB      CR,LF,'0062    BRKPT',TAB,'BREAKPOINT ROUTINE'
DB      04H
END

```


August

density.src

Programming Insight: "Polar Normal Distribution," by
Alain Latour. August, page 131. Also see normal.src.

```
function NormDens (x : real) : real;
  const c = 0.3989422804;
  begin
    NormDens := c * exp(-sqr(x) / 2)
  end;

function NormProb (x : Real) : Real;
  const b0 = +0.319381530; b1 = -0.356563782;
        b2 = +1.781477937; b3 = -1.821255978;
        b4 = +1.330274429; p = +0.2316419;
  var Temp, t : Real;
      Positive : Boolean;

  begin
    Positive := x > 0;
    x := Abs(x);
    t := 1 / (1 + p * x);
    Temp := NormDens(x)*t*(b0+t*(b1+t*
      (b2+t*(b3+b4*t))));
    if Positive then Temp := 1 - Temp;
    NormProb := Temp
  end;
```

normal.src

Programming Insight: "Polar Normal Distribution," by
Alain Latour. August, page 131. Also see density.src.

Program Normal;

```
Var I : Integer;
    Y1, Y2 : Real;
    Freq : Array[0..30] of Integer;
```

Procedure Initialize;

```
Var I : integer;
Begin
  For i:= 0 to 30 do Freq[i] := 0
End;
```

Procedure Classify (Y: Real);

```
Const MinY = -3.5;
      YRange = 7.0;
      NbClasses = 30;
Var Temp : Integer;
Begin
  Temp := Trunc((Y-MinY)/YRange*NbClasses);
  If Temp < 1 then Temp := 0
    else if Temp > NbClasses then Temp :=
      NbClasses;
  Freq[Temp] := Freq[Temp] + 1
End;
```

Procedure NorDev (Var Y1, Y2 : Real);

```
Var V1, V2, S : Real;
Begin
  { The "Repeat until" loop is repeated 1.27 times
```

(continued)

on the average with a standard deviation
of 0.587 (c.f. D. Knuth (1969), page 104) }

```
Repeat
  V1 := 2*Random -1;
  V2 := 2*Random -1;
  S := Sqr(V1) + Sqr(V2)
  until S < 1;
  S := Sqrt(-2*ln(S)/S);
  Y1 := V1*S;
  Y2 := V2*S
End;

Begin
  Initialize;
  For i:= 1 to 5000 do
    Begin
      NorDev(Y1,Y2);
      Classify(Y1);
      Classify(Y2)
    End;
  For i:=0 to 30 do
    Writeln(I:6,Freq[I]:12)
  End.
```

lets.src

"Let's C and CSD," by William Wong. August, page 267.

```
/* ==== Byte Benchmark: Eratosthenes Sieve
      02-22-86  WGW  ==== */

#define ITERATIONS 10 /* number of times to
                        perform test */

#define LOCAL 1 /* use stack for flags */
#define GLOBAL 0 /* use global area for flags */

typedef char flag ; /* different types make a
                    difference */

#include <stdio.h> /* standard I/O definitions */

/* ---- Main Function ---- */

#define SIZE 7000

#if GLOBAL
flag flags [ SIZE + 1 ] ; /* prime number flag array */
#endif

main ()
{
  /* ---- Variable definitions ---- */
  /*
  /* Note: only some register specifications will
  be used due to the */
  /* limited number of registers available.
  Normally, the first */
  /* items are allocated to registers. */

  register int i ; /* ordered by preference */
  register int k ;
  register int prime ;
  register int count ;
  register int iterations ;

  #if LOCAL
    flag flags [ SIZE + 1 ] ;
    /* prime number flag array */
  #endif
```



```

/* ---- Show off start of execution ---- */
printf ( "Starting Eratosthenes Sieve
          Benchmark\n\n" );
printf ( "%d iterations.\n\n", ITERATIONS );

for ( iterations = ITERATIONS ; iterations ; --
      iterations )
{
    /* ---- Initialize Sieve Flag Array ---- */

    for ( i = 0 ; i < SIZE ; ++ i )
        flags [ i ] = 1 ; /* mark all as prime numbers */

    /* ---- Search for Prime Numbers ---- */

    for ( i = 0 /* scan for next prime number */
          ; count = 0 /* keep count of primes found */
          ; i < SIZE
          ; ++ i
          )
    {
        if ( flags [ i ] != 0 ) /* check if prime found */
        {
            /* ---- Prime found. Count it and unmark
              multiples ---- */

            ++ count ; /* increment number of primes */

            for ( prime = i + i + 3
                  /* mark multiples as non-prime */
                  , k = i + prime
                  /* start with first multiple */
                  ; k < SIZE
                  ; k += prime
                  )
                flags [ k ] = 0 ;
            /* reset multiples to non-prime */
        }
    }

    /* ---- Mark end of execution ---- */

    printf ( "%d prime numbers found.\n", count ) ;

    printf ( "End of Test\n\n" ) ;
}

/* ===== End of Eratosthenes Sieve Benchmark ===== */
.pa
/* ===== Byte Benchmark: Calculation Test 02-22-86 WGW ===== */

#include <stdio.h> /* standard I/O definitions */

/* ===== Main Function ===== */
#define ITERATIONS 5000

main ()
{
    float a, b, c ;
    int i ;

    /* ---- Show start of test ---- */

    printf ( "Start of Calculation Test.\n\n" ) ;

    /* ---- Perform test loop ---- */

```

(continued)

```

for ( i = ITERATIONS /* setup for timing loop */
    , a = 2.71828
    , b = 3.14159
    , c = 1
    ; i
    ; -- i
    )
{
/* ---- Perform balanced set of calculations ---- */

    c *= a ; /* same as c = c * a */
    c *= b ;

    c /= a ; /* same as c = c / a */
    c /= b ;
}

/* ---- Show end of test ---- */

printf ( "End of test. Accumulated error:
        %f\n\n", c - 1 ) ;
}

/* ==== End of Eratosthenes Sieve Benchmark ==== */
.pa
/* ==== Byte Benchmark: Write 64 kbyte file
        02-22-86 WGW ==== */

#include <stdio.h> /* standard I/O definitions */

/* ==== Main Test Function ==== */

#define RECORDS 512 /* records in test file */
#define REC_SIZE 128 /* size of records,
                    total is 64kbytes */

main ()
{
    int file ;
    int records ;
    char buffer [ REC_SIZE ] ;
    int i ; /* buffer fill index */

static char file_name [] = "B:TEST" ;

/* ---- Show start of test ---- */

printf ( "Writing TEST file.\n\n" ) ;

/* ---- Fill output buffer with recognizable
        information ---- */

for ( i = 0 ; i < sizeof ( buffer ) ; ++ i )
    buffer [ i ] = i ;

/* ---- Write information ---- */

if ( ( file = creat ( file_name, 0 ) ) > -1 )
{
/* ---- File opened, try writing all the records ---- */

for ( records = RECORDS ; records ; -- records )
    if ( write ( file, buffer, sizeof ( buffer ) ) !=
        sizeof ( buffer ) )
    {
        printf ( "Write error.\n" ) ; /* show error */
        break ; /* exit from loop */
    }
}
}

```



```

/* ---- Close down the file ---- */
close ( file ) ;
}
else
    printf ( "Cannot open %s\n", file_name ) ;

/* ---- Show end of test ---- */
printf ( "End of Write Test.\n\n" ) ;
}

/* ==== End of Write 64 kbyte File Benchmark ==== */

.pa

/* ==== Byte Benchmark: Read 64 kbyte file
02-22-86 WGW ==== */

#include <stdio.h> /* standard I/O definitions */

/* ==== Main Test Function ==== */
#define RECORDS 512 /* records in test file */
#define REC_SIZE 128 /* size of records,
total is 64kbytes */

main ()
{
    int file ;
    int records ;
    char buffer [ REC_SIZE ] ;
    static char file_name [] = "B:TEST" ;

/* ---- Show start of test ---- */
printf ( "Reading TEST file.\n\n" ) ;

/* ---- Read information ---- */
if ( ( file = open ( file_name, 0 ) ) > -1 )
{
    /* ---- File opened, try reading all the records ---- */
    for ( records = RECORDS ; records ; -- records )
        if ( read ( file, buffer, sizeof ( buffer ) ) !=
            sizeof ( buffer ) )
        {
            printf ( "Read error.\n" ) ; /* show error */
            break ; /* exit from loop */
        }

/* ---- Close down the file ---- */
close ( file ) ;
}
else
    printf ( "Cannot open %s\n", file_name ) ;

/* ---- Show end of test ---- */
printf ( "End of Read Test.\n\n" ) ;
}

/* ==== End of Read 64 kbyte File Benchmark ==== */

```

macapp.use

"MacApp: An Application Framework," by Kurt J. Schmucker. August, page 189.

```

                                { The Smallest MacApp Application }
                                { Copyright 1986 by Productivity Products International, Inc.}

{ The MAIN Program }
PROGRAM SmallApp;

USES
  { This set of units are portions of the Macintosh ROM }
  MemTypes, QuickDraw, OSIntf, ToolIntf,

  { This set of units are portions of MacApp }
  UObject, UList, UMacApp, UPrinting,

  { This unit has the SmallApp-specific classes }
  USmallApp;

VAR aSmallApplication: TSmallApplication; { The application object -
only one of these per application }

BEGIN
  InitToolbox(8);      { initialize the ToolBox; 8 calls to MoreMasters }
  InitPrinting;        { initialize the print shop }

  New(aSmallApplication);
  aSmallApplication.ISmallApplication;
  aSmallApplication.Run;
END.

{ *** The Unit it uses. This is typically in two separate files: one for the
interface and one for the implementation. ***}

                                { The Smallest Possible MacApp Application }
                                { Copyright 1986 by Productivity Products International,
Inc. }

UNIT USmallApp;

INTERFACE

USES
  { This set of units are portions of the Macintosh ROM }
  MemTypes, QuickDraw, OSIntf, ToolIntf, PackIntf,

  { This set of units are portions of MacApp }
  UObject, UList, UMacApp, UPrinting;

CONST
  myFileType = 'MAMO';      { The file type ("MacApp MUse") for documents
of this application }
  mySignature = 'SMAP';      { The application signature of Small
Application }
  myWindowType = 1001;      { The resource ID of the WIND resource which
                             defines the windows used to
                             display the documents of this application.
}

TYPE

  TSmallApplication = OBJECT(TApplication)

    { ----- INITIALIZE THE APPLICATION ----- }
    PROCEDURE TSmallApplication.ISmallApplication;

    { ----- MAKE A DOCUMENT ----- }
    FUNCTION TSmallApplication.DoMakeDocument(itsCmdNumber: CmdNumber):
TDocument; OVERRIDE;

```


END;

TSmallDocument = OBJECT(TDocument)

```

    { ----- FIELDS ----- }
    fSmallView: TSmallView;

    { ----- INITIALIZE A DOCUMENT ----- }
    PROCEDURE TSmallDocument.ISmallDocument;

    { ----- MAKE A VIEW ----- }
    PROCEDURE TSmallDocument.DoMakeViews(forPrinting: BOOLEAN); OVERRIDE;

    { ----- MAKE A WINDOW ----- }
    PROCEDURE TSmallDocument.DoMakeWindows; OVERRIDE;

```

END;

TSmallView = OBJECT(TView)

```

    { ----- INITIALIZE A VIEW ----- }
    PROCEDURE TSmallView.ISmallView(itsSmallDocument: TSmallDocument);

    { ----- RENDER THE IMAGE ----- }
    PROCEDURE TSmallView.Draw(area: Rect); OVERRIDE;

```

END;

IMPLEMENTATION

{ Copyright 1985 by Productivity Products International, Inc.}

{ USmallApp Implementation }

```

{
*****
*****
*****
METHODS FOR ALL THE SMALLAPP CLASSES

Note that methods are grouped by class and that the order of methods in
any class is the following }
(by convention only, since Object Pascal forces no order): }

    (1) the initialization method, if any, }
    (2) the Inspect method - a private debugging method, if needed }
    (3) the Free method, if overridden, and}
    (4) the remaining methods in alphabetical order. }

*****
*****

```

{ ***** TSmallApplication Methods ***** }

```

PROCEDURE TSmallApplication.ISmallApplication;
BEGIN
    SELF.IApplication(myFileType);
END;

{ ----- MAKE AND INITIALIZE A DOCUMENT ----- }

FUNCTION TSmallApplication.DoMakeDocument(itsCmdNumber: CmdNumber):
    TDocument; OVERRIDE;
VAR aSmallDocument: TSmallDocument;

```

(continued)

```

BEGIN
    NEW(aSmallDocument);
    aSmallDocument.ISmallDocument;
    DoMakeDocument := aSmallDocument;
END;

{ ***** TSmallDocument Methods ***** }

PROCEDURE TSmallDocument.ISmallDocument;
BEGIN
    SELF.IDocument(myFileType, mySignature, TRUE, FALSE);
END;

{ ----- MAKE AND INITIALIZE ALL THE NECESSARY VIEWS ----- }

PROCEDURE TSmallDocument.DoMakeViews(forPrinting: BOOLEAN); OVERRIDE;
VAR smallView: TSmallView;
BEGIN
    NEW(smallView);
    smallView.ISmallView(SELF);
    SELF.fSmallView := smallView;
END;

{ ----- MAKE ALL THE NECESSARY WINDOWS ----- }

PROCEDURE TSmallDocument.DoMakeWindows; OVERRIDE;
VAR aWindow: TWindow;
BEGIN
    aWindow := NewSimpleWindow(myWindowType, FALSE { NOT a DialogWindow},
                               kWantHScrollBar, kWantVScrollBar,
                               SELF.fSmallView);
END;

{ ***** TSmallView Methods ***** }

PROCEDURE TSmallView.ISmallView(itsSmallDocument: TSmallDocument);
VAR viewRect: Rect;
    aStdHandler: TStdPrintHandler;
BEGIN
    SetRect(viewRect, 0, 0, 500, 500);
    IView(NIL,
           itsSmallDocument,
           viewRect,
           sizeFixed,
           sizeFixed,
           FALSE,
           hIOff);
    { This view has no parent view,
      and shows a smallDocument,
      in a 500 x 500 rectangle,
      that does not change if the
      frame is changed horizontally,
      or vertically,
      and can't make selections
      and doesn't highlight when the
      window is inactive. }

    New(aStdHandler);
    aStdHandler.IStdPrintHandler(SELF, FALSE); { The second parameter,
    itsSquareDots, is FALSE since this application does not mix text
    and graphics. Slightly higher resolution is available with this
    setting. }
END;

{ ----- RENDER THE IMAGE ----- }

PROCEDURE TSmallView.Draw(area: Rect); OVERRIDE;

FUNCTION MakeRect(top, left, bottom, right: INTEGER): Rect;
VAR r: Rect;
BEGIN
    SetRect(r, left, top, right, bottom);
    MakeRect := r;
END;

```



```

BEGIN
  PenNormal;
  PaintOval(MakeRect(74, 72, 139, 127));
  EraseOval(MakeRect(84, 74, 138, 125));
  FrameOval(MakeRect(109, 84, 129, 115));
  EraseRect(MakeRect(109, 84, 123, 115));
  FrameOval(MakeRect(98, 87, 107, 96));
  FrameOval(MakeRect(98, 104, 107, 113));
  PaintOval(MakeRect(101, 90, 104, 93));
  PaintOval(MakeRect(101, 107, 104, 110));
  PaintOval(MakeRect(111, 97, 117, 103));
  PaintOval(MakeRect(53, 52, 91, 90));
  PaintOval(MakeRect(53, 110, 91, 148));

  FrameRect(MakeRect(20, 20, 170, 180));

  { Outline of the mouse head }
  { Outline of the mouse face }
  { Mouse mouth (part 1 of 2) }
  { Mouse mouth (part 2 of 2) }
  { Left eye }
  { Right eye }
  { Left pupil }
  { Right pupil }
  { Nose }
  { Left ear }
  { Right ear }

  { A bounding rectangle }

END;

END.

```

mapper.bas

"Similarity Mapping," by Rob Spencer. August, page 85.

```

' similarity mapping
'   I Robin W. Spencer

```

```

GOSUB Initialize
GOSUB SetMenus
GOSUB InputData
GOSUB ShowTable
GOSUB InitializeMap
GOSUB ShowMap
GOSUB RefineMap
GOSUB ShowMap

```

```

WHILE NOT AllDone
  GOSUB WaitForMenu
  GOSUB HandleMenu
  GOSUB ShowMap
WEND

```

```

GOSUB ShowQuitBox
IF ButtonId=1 THEN RUN ELSE SYSTEM

```

```

'
'                                     subroutines

```

```

Initialize:
  DEFINT i-n
  WINDOW 1,,(0,20)-(512,340),3
  CALL TEXTFONT(0):CALL TEXTSIZE(12)
  CLS

  True = -1 : False = 0
  AllDone=False
  MaxIterations=20
  R = .15 : Criterion = .2

  xleft = 0 : xright = 512   ' screen window in pixels
  ytop = 20 : ybottom = 320

  xcenter = (xleft + xright)/2
  ycenter = (ytop + ybottom)/2
  DEF FNPx(x) = x*Scale + xcenter
  DEF FNPy(y) = y*Scale + ycenter
  RETURN

```

```

SetMenus:
  MENU 1,0,1,"Control"
  MENU 1,1,1,"Change Map"

```

(continued)

```

MENU 1,2,1,"Show Table"
MENU 1,3,0,"-"
MENU 1,4,1,"Quit"

MENU 2,0,0,""
MENU 3,0,0,""
MENU 4,0,0,""
MENU 5,0,0,""
RETURN

```

InputData:

```

PRINT:PRINT"  Select the input file:"
InputFile$ = FILES$(1,"TEXT")
OPEN InputFile$ FOR INPUT AS #1

INPUT#1, Title$
INPUT#1, NumPoints
  NumPairs = NumPoints*(NumPoints-1)/2
DIM
d(NumPoints,NumPoints),x(NumPoints),y(NumPoints),Dx(NumPoints),Dy(NumPoints),L
abel$(NumPoints)
FOR i=1 TO NumPoints
  INPUT#1, Label$(i)
NEXT i
FOR i=2 TO NumPoints
  FOR j=1 TO i-1
    INPUT#1, d(i,j)
    d(j,i)=d(i,j)
    IF d(i,j)<0 THEN NumPairs = NumPairs-1
  NEXT j
NEXT i
CLOSE#1
RETURN

```

InitializeMap:

```

dmax=0
find the two most distant points
FOR i=1 TO NumPoints-1
  FOR j=i+1 TO NumPoints
    IF d(i,j)>dmax THEN dmax=d(i,j) : j1=i : j2=j
  NEXT j
NEXT i

x(j1)=0 : y(j1)=0
x(j2)=dmax : y(j2)=0
Scale = .6*(xright-xleft)/dmax ' scale in pixels per distance unit

dnext=0
find the third most distant point
FOR i=1 TO NumPoints
  IF i=j1 OR i=j2 THEN FirstLoop
  delta=d(i,j1)+d(i,j2)-d(j1,j2)
  IF delta>dnext THEN dnext=delta:j3=i
FirstLoop:
NEXT i

x(j3)=d(j1,j3)-dnext/2
y(j3)=SQR(d(j1,j3)^2-x(j3)^2)

FOR i=1 TO NumPoints
  initialize all the other points
  IF i=j1 OR i=j2 OR i=j3 THEN SecondLoop
  x(i)=(d(j1,i)^2-d(j2,i)^2+dmax^2)/(2*dmax)
  y(i)=SQR(ABS(d(j1,i)^2-x(i)^2))
  IF (x(i)-x(j3))^2+y(j3)^2 < d(j3,i)^2 THEN y(i)=-y(i)
SecondLoop:
NEXT i

FOR i=1 TO NumPoints
  the coordinate set
  x(i) = x(i) - dmax/2
NEXT i
RETURN

```

center

RefineMap:

```

Converged = False : Iteration = 1 : OldError = 10000

```



```

WHILE NOT AllDone AND NOT Converged AND Iteration < MaxIterations
  GOSUB IterateMap
  IF MENU(0)>0 THEN GOSUB HandleMenu
  GOSUB ShowMap
  IF ABS(PercentError - OldError) < Criterion THEN Converged = True
  IF PercentError > OldError THEN R = .7*R
  OldError = PercentError
  Iteration = Iteration + 1
WEND
Title$ = Title$ + " (final)"
RETURN

IterateMap:
  SumError=0
  FOR i=1 TO NumPoints
    Dx(i)=0 : Dy(i)=0
  NEXT i

  FOR i=1 TO NumPoints-1      ' calculate the changes Dx() and Dy()
    FOR j=i+1 TO NumPoints
      IF d(i,j)<0 THEN InnerLoop      ' allow for unknown
distances
        dcalc=SQR((x(i)-x(j))^2 + (y(i)-y(j))^2)
        IF dcalc=0 THEN InnerLoop
        delta = (d(i,j)/dcalc-1)*(x(j)-x(i))
        Dx(i)=Dx(i)-delta
        Dx(j)=Dx(j)+delta
        delta = (d(i,j)/dcalc-1)*(y(j)-y(i))
        Dy(i)=Dy(i)-delta
        Dy(j)=Dy(j)+delta
        SumError = SumError + ((dcalc-d(i,j))/d(i,j))^2
      InnerLoop:
    NEXT j
  NEXT i

  FOR i=1 TO NumPoints      ' make the changes to the points
    x(i)=x(i) + Dx(i)*R
    y(i)=y(i) + Dy(i)*R
  NEXT i

  PercentError = 100*SQR(SumError/NumPairs)
  RETURN

WaitForMenu:
  WHILE MENU(0)=0
  WEND
  RETURN

HandleMenu:
  ItemId = MENU(1)
  IF ItemId = 4 THEN AllDone=True : RETURN
  ON ItemId GOSUB ShowPanel,ShowTable
  MENU 1,0,1
  RETURN

ShowTable:
  CALL TEXTFONT(4):CALL TEXTSIZE(9): CALL TEXTMODE(1)
  CLS
  PRINT:PRINT" Distance table for ";Title$
  IF Iteration>0 THEN GOSUB SetFormat ELSE Format$=" #####"
  PRINT
  FOR i=1 TO NumPoints
    PRINT TAB(12+6*i);i;
  NEXT i
  PRINT:PRINT
  Sum=0
  FOR i=1 TO NumPoints
    PRINT USING"###) \      \";i,Label$(i);
    FOR j=1 TO NumPoints
      IF j<i THEN d=d(i,j) ELSE d=0
      IF j>i AND Iteration>0 THEN d=SQR((x(i)-x(j))^2 + (y(i)-y(j))^2)
      IF j>i AND d(i,j)>0 THEN Sum=Sum + (d-d(i,j))^2
      IF d>0 THEN PRINT USING Format$;d; : ELSE PRINT SPACE$(6);
    NEXT j
  PRINT

```

(continued)

```

NEXT I
PRINT:PRINT TAB(40);"observed \ calculated":PRINT
IF Iteration>0 THEN PRINT USING " Average error = "+Format$+" (###.## %)";
SQR(Sum/NumPairs),PercentError
BUTTON 1,1,"Ok",(400,280)-(480,310)
GOSUB GetButton
BUTTON CLOSE 1
RETURN

```

```

SetFormat:
length=INT(.43*LOG(ABS(dmax)) + 1.5)
Format$=" "+STRING$(length,"#")+ "."
WHILE LEN(Format$)<6
  Format$=Format$+"#"
WEND
RETURN

```

```

ShowMap:
CLS
PRINT Title$
PRINT USING " Iteration ##";Iteration
IF Iteration>0 THEN PRINT USING " ###.## % error";PercentError
FOR i=1 TO NumPoints
  CALL MOVETO(FNPx(x(i)),FNPy(y(i)))
  PRINT "E ";Label$(i);
NEXT I
RETURN

```

```

ShowPanel:
WINDOW 2,,(3,24)-(509,48),-4
CALL TEXTFONT(0):CALL TEXTSIZE(12)
BUTTON 1,1,"rotate",(2,2)-(52,22)
EDIT FIELD 1,"0",(56,4)-(88,19)
RESTORE ShowPanel
x=94
FOR i=2 TO 8
  READ Id$
  BUTTON i,1,Id$,(x,2)-(x+52,22)
  x=x+54
NEXT i
DATA expand,shrink,flip,up,down,left,right
BUTTON 9,1,"Ok",(x,2)-(x+30,22)

WHILE DIALOG(0)<>0:WEND ' flush pending dialog events
Done=False
WHILE NOT Done
  GOSUB GetButton
  GOSUB MoveMap
  WINDOW OUTPUT 1
  GOSUB ShowMap
  WINDOW OUTPUT 2
WEND
WINDOW CLOSE 2
RETURN

```

```

GetButton:
Event=0
WHILE Event<>1
  Event = DIALOG(0)
WEND
ButtonId=DIALOG(1)
RETURN

```

```

MoveMap:
IF ButtonId=1 THEN GOSUB RotateMap
IF ButtonId=2 THEN Scale=Scale*1.1
IF ButtonId=3 THEN Scale=Scale/1.1
IF ButtonId=4 THEN GOSUB FlipMap
IF ButtonId=5 THEN ycenter = ycenter - 5
IF ButtonId=6 THEN ycenter = ycenter + 5
IF ButtonId=7 THEN xcenter = xcenter - 5
IF ButtonId=8 THEN xcenter = xcenter + 5
IF ButtonId=9 THEN Done=True
RETURN

```



```

RotateMap:
  theta = VAL(EDIT$(1))
  theta=theta*3.14159/180
  sine=SIN(theta) : cosine=COS(theta)
  FOR i=1 TO NumPoints
    xt = x(i)*cosine - y(i)*sine
    y(i)= y(i)*cosine + x(i)*sine
    x(i)= xt
  NEXT i
  RETURN

FlipMap:
  FOR i=1 TO NumPoints
    x(i)= -x(i)
  NEXT i
  RETURN

ShowQuitBox:
  WINDOW 2,"",(150,105)-(360,160),-2
  BUTTON 1,1,"Run Again",(10,10)-(100,40)
  BUTTON 2,1,"Quit",(110,10)-(200,40)
  GOSUB GetButton
  WINDOW CLOSE 2
  CLS
  RETURN

```

micros.dat

"Similarity Mapping," by Rob Spencer. August, page 85.

"Cluster Example"
12

Ace,Apple IIe,Atari 800,Comm 64,Compaq,IBM PC,Kaypro II,Osborne I,TI 99/4A,TRS Color/80-4,Vic 20

2.66																			
3.79	2.61																		
3.94	2.82	1.17																	
5.41	5.46	6.79	6.34																
4.15	4.20	5.23	4.82	3.26															
2.49	4.07	4.87	4.98	5.33	5.19														
2.43	3.57	4.37	4.47	5.01	4.85	0.99													
4.33	3.34	1.10	1.88	7.41	5.66	5.28	4.80												
4.18	3.15	1.02	1.85	7.30	5.52	5.17	4.69	0.24											
2.08	2.12	3.42	3.33	4.70	3.27	3.63	3.03	3.99	3.84										
4.69	3.80	1.83	1.96	7.36	5.46	5.58	5.13	1.13	1.19	4.19									

cities.dat

"Similarity Mapping," by Rob Spencer. August, page 85.

"Intercity distances"

9
Boston,NY,DC,Miami,Chicago
Seattle,SF,LA,Denver
206
429, 233

1504,1308,1075
963, 802, 671,1329
2976,2815,2684,3273,2013
3095,2934,2799,3053,2142, 808
2979,2786,2631,2687,2054,1131, 379
1949,1771,1616,2037, 996,1307,1235,1059

cytoc.dat

"Similarity Mapping," by Rob Spencer. August, page 85.

"Cytochrome C sequence distances"

12
human,monkey,dog,horse,donkey
pig,rabbit,kangaroo,duck
pigeon,chicken,turtle
1
13,12

17,16,10
16,15, 8, 1
13,12, 4, 5, 4
12,11, 6,11,10, 6
12,13, 7,11,12, 7, 7

(continued)

17,16,12,16,15,13,10,14
16,15,12,16,15,13, 8,14, 3

18,17,14,16,15,13,11,15, 3, 4
19,18,13,16,15,13,11,14, 7, 8, 8

factsum.bas

Mathematical Recreations:"Number Games," by Robert T. Kurosaka. August, page 333.

```

10 REM -----< Sum-of-Factorials Routine >-----
20 REM -----<           Bob Kurosaka           >-----
30 REM
40 DIM A(10),B(100),F(10):REM A(1)=digits, B(1)=sums,
   F(1)=factorials.
50 DATA 1,1,2,6,24,120,720,5040,40320,362880
60 FOR J=1 TO 10
70 READ F(J)
80 NEXT J
90 CLS
100 PRINT "Program generates factorial sum
   sequences."
110 PRINT
120 PRINT "Lower limit, upper limit ";
130 INPUT LL, UL
140 LL=ABS(INT(LL))
150 UL=ABS(INT(UL))
160 REM
170 FOR N=LL TO UL:REM Sequences for each no.
   LL to NN
180 SP=0:REM SP counts the steps
   before a cycle
190 B(SP)=N:REM Make the first term=N
200 PRINT B(SP):REM Print the current term
210 M=B(SP):REM Make a copy of latest term
220 REM
230 REM Break up the term into its component digits
240 D=1:REM D = no. of digits
250 T=INT(M/10):REM T = no. of "Tens" in M
260 A(D)=M-10*T:REM Store rightmost digit
   in array A
270 IF T<>0 THEN D=D+1: M=T: GOTO 250
280 REM
290 REM Calculate the sum of the factorials of
   the digits in A(1)
300 SUM=0
310 FOR I=1 TO D
320 SUM=SUM+F(A(I)+1)
330 NEXT I
340 REM See if sum has occurred already.
350 I=0
360 WHILE B(I)<>SUM AND I<=SP
370 I=I+1
380 WEND
390 IF B(I)=SUM AND I<=SP THEN 440
400 SP=SP+1:REM one more step
410 B(SP)=SUM:REM Store SUM in array B
420 GOTO 200
430 REM
440 PRINT SUM; "* loops to step "; I:REM Show
   where it repeats
450 PRINT
460 NEXT N
470 END

```

factsum.run

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

Program generates factorial sum sequences.

Lower limit, upper limit ? 24, 27

24 26 722 5044 169 363601 1454 169 *
loops to step 4

25 122 5 120 4 24 26 722 5044 169 363601
1454 169 * loops to step 9

26 722 5044 169 363601 1454 169 * loops
to step 3

27 5042 147 5065 961 363601 1454 169
363601 * loops to step 5

Ok

cubesum.bas

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

```

10 REM -----< Sum-of-Cubes Routine >-----
20 REM -----<      Bob Kurosaka      >-----
30 REM
40 DIM A(10),B(100)      :REM A() holds digits,
                        :REM B() holds sums.
50 CLS
60 PRINT "Program generates sequences of
  cube-sums."
70 PRINT
80 PRINT "Lower limit, upper limit ";
90 INPUT LL, UL
100 LL=ABS(INT(LL))
110 UL=ABS(INT(UL))
120 REM
130 FOR N=LL TO UL      :REM Sequences for each
                        :REM no. LL to NN
140 SP=0                :REM SP counts the steps
                        :REM before a cycle
150 B(SP)=N              :REM Make the first term=N
160 PRINT B(SP);        :REM Print the current
                        :REM term
170 M=B(SP)             :REM Make a copy of
                        :REM latest term
180 REM
190 REM Break up the term into its
  component digits
200 D=1                  :REM D = no. of digits
210 T=INT(M/10)          :REM T = no. of "Tens" in M
220 A(D)=M-10*T          :REM Store rightmost digit in array A
230 IF T<>0 THEN D=D+1: M=T: GOTO 210
240 REM
250 REM Calculate the sum of the cubes of the
  digits in A()
260 SUM=0
270 FOR I=1 TO D
280 SUM=SUM+A(I)*A(I)*A(I)
290 NEXT I
300 REM See if sum has occurred already.
310 I=0
320 WHILE B(I)<>SUM AND I<=SP
330 I=I+1
340 WEND
350 IF B(I)=SUM AND I<=SP THEN 400

```

(continued)

August

```
360 SP=SP+1      :REM one more step
370 B(SP)=SUM     :REM Store SUM in array B
380 GOTO 160
390 REM
400 PRINT SUM; "* loops to step "; I :REM Show
    where it repeats
410 NEXT N
420 END
```

cubesum.run

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

Program generates sequences of cube-sums.

Lower limit, upper limit ? 21, 25
21 9 729 1080 513 153 153 *
 loops to step 5

22 16 217 352 160 217 * loops to step 2

23 35 152 134 92 737 713 371 371 *
 loops to step 7
24 72 351 153 153 * loops to step 3
25 133 55 250 133 * loops to step 1
Ok

palind.bas

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

```
10 REM -----< Palindromic Sums Routine >-----
20 REM -----<      Bob Kurosaka      >-----
30 REM
40 SP$=" "      :REM One space inside quotes
50 D$="0123456789"
60 YES=(1=1)
70 DIM A(100)    :REM A() holds the the digits.
80 CLS
90 PRINT "Program generates sequences that end
    in palindromes."
100 PRINT
110 PRINT "Lower limit (>10) and upper limit ";
120 INPUT LL, UL
130 LL=ABS(INT(LL))
140 UL=ABS(INT(UL))
150 IF LL<10 THEN 110
160 FOR N=LL TO UL
170 SP=0      :REM SP counts the steps before a cycle
180 REM
190 REM Break up the term into its component digits
200 M=N      :REM Make a copy of latest term
210 D=1      :REM D = no. of digits
220 T=INT(M/10) :REM T = no. of "Tens" in M
230 A(D)=M-10*T :REM Store rightmost digit in
    array A
240 IF T<>0 THEN D=D+1: M=T: GOTO 220
250 ODD=ABS((INT(D/2)<>D/2)) :REM Even or odd
    no. of digits?
260 REM
270 REM Print the latest term
280 FOR I=D TO 1 STEP -1
290 PRINT MID$(D$,A(I)+1,1);
300 NEXT I
310 PRINT SP$;
320 REM
330 REM Check for palindrome
340 FOR I=1 TO D/2
350 PL=(A(I)=A(D-I+1))
360 IF NOT PL THEN I=D/2 :REM Exit from loop if
    no. is not a pal.
370 NEXT I
380 IF PL THEN 580
390 REM
400 REM Add each digit to its reverse image
    counterpart
```



```

410 FOR I=1 TO D/2+ODD
420 A(I)=A(I)+A(D-I+1)
430 A(D-I+1)=A(I)
440 NEXT I
450 REM Check for carry
460 FOR I=1 TO D
470 IF A(I)<10 THEN 500
480 A(I)=A(I)-10
490 A(I+1)=A(I+1)+1
500 NEXT I
510 IF A(D+1)=0 THEN 540
520 D=D+1
530 ODD=ABS((INT(D/2)<>D/2))
540 SP=SP+1
550 GOTO 280
560 REM
570 REM Indicate that a cycle has been found
580 PRINT "* at step "; SP
590 FOR I=1 TO D
600 A(I)=0
610 NEXT I
620 NEXT N
630 END

```

palind.run

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

Program generates sequences that end in palindromes.

Lower limit (>10) and upper limit ? 75, 80
 75 132 363 * at step 2
 76 143 484 * at step 2
 77 * at step 0
 78 165 726 1353 4884 * at step 4

79 176 847 1595 7546 14003 44044 * at
 step 6
 80 88 * at step 1
 Ok

powser.bas

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

```

10 REM -----< Delete and Add Routine >-----
20 REM -----<      Bob Kurosaka      >-----
30 REM
40 N=50      :REM N=number of integers in starting
              sequence.
50 DIM SF$(3), A(N) :REM SF$() holds suffixes,
  A() holds sequence.
60 DATA nd, rd, th
70 FOR J=1 TO 3: READ SF$(J): NEXT J
80 REM
90 CLS
100 PRINT "Program uses a delete-and-add process"
110 PRINT "to generate i^p for i=1 to ";N; "/" p."
120 PRINT: INPUT "Enter a value for p (power)"; P
130 P=ABS(INT(P))
140 IF P<2 THEN 100
150 REM
160 PRINT: PRINT "Starting sequence:"
170 FOR I=1 TO N
180 A(I)=I
190 PRINT I;
200 NEXT I
210 PRINT
220 REM
230 FOR R=P TO 2 STEP-1

```

(continued)

August

```
240 DC=0                :REM Counts the number
                        :REM of terms deleted.
250 FOR J=R TO N STEP R :REM Delete every Jth term
260 A(J)=0
270 DC=DC+1
280 NEXT J
290 REM
300 REM -----< Print deleted array >-----
310 WSF=SGN(R-3)+2      :REM Select suffix
320 PRINT: PRINT "Delete every "; R; SF$(WSF);
  " term:"
330 FOR I=1 TO N
340 IF A(I)<>0 THEN PRINT A(I); ELSE PRINT "*";
350 NEXT I
360 REM
370 REM -----< Compute Partial Sums >-----
380 K=1
390 FOR J=2 TO N-DC      :REM There will be N-DC
                        :REM valid numbers.
400 K=K+1                :REM K points to next
                        :REM term to be added.
410 IF A(K)=0 THEN K=K+1 :REM Skip zero (deleted)
                        :REM terms.
420 A(J)=A(J-1)+A(K) :REM Calculate partial sum.
430 NEXT J
440 N=N-DC              :REM Revise the number of valid
                        :REM terms in A().
450 REM
460 REM -----< Print Partial Sums >-----
470 PRINT: PRINT: PRINT "Partial sums:"
480 FOR I=1 TO N
490 PRINT A(I);
500 NEXT I
510 PRINT
520 NEXT R
530 END
```

powser.run

Mathematical Recreations: "Number Games," by Robert T. Kurosaka. August, page 333.

Program uses a delete-and-add process
to generate i^p for $i=1$ to $50 / p$.

Enter a value for p (power)? 4

Starting sequence:

1	2	3	4	5	6	7	8	9	10	11	12	13
14	15	16	17	18	19	20	21	22	23	24		
25	26	27	28	29	30	31	32	33	34	35		
36	37	38	39	40	41	42	43	44	45	46		
47	48	49	50									

Delete every 4 th term:

1	2	3	* 5	6	7	* 9	10	11	* 13	14	15	*
17	18	19	*	21	22	23	* 25	26	27	* 29	30	
31	* 33	34	35	* 37	38	39	* 41	42	43	* 45		
46	47	* 49	50									

Partial sums:

1	3	6	11	17	24	33	43	54	67	81	96	
113	131	150	171	193	216	241	267	294				
323	353	384	417	451	486	523	561	600	641			
683	726	771	817	864	913	963						

Delete every 3 rd term:

1	3	* 11	17	* 33	43	* 67	81	* 113	131	*
171	193	* 241	267	* 323	353	* 417	451	*		
523	561	* 641	683	* 771	817	* 913	963			

Partial sums:

1	4	15	32	65	108	175	256	369	500			
671	864	1105	1372	1695	2048	2465	2916					
3439	4000	4641	5324	6095	6912	7825	8788					

Delete every 2 nd term:

```
1 * 15 * 65 * 175 * 369 * 671 * 1105 * 1695 *
2465 * 3439 * 4641 * 6095 * 7825 *
```

Partial sums:

```
1 16 81 256 625 1296 2401 4096 6561
10000 14641 20736 28561
```

Ok

readfort.me

"Object-Oriented FORTH," by Dick Pountain, August, page 227. Also see types.doc.

A minor addition concerning error handling must be made to the TYPES.SCR or TYPES.DOC code. In screen 2 of the source code, the system sets up a third stack called OSTACK, which incorporates no checking for underflow or overflow. In certain circumstances, overflow could occur. If you are debugging a new type and an operation crashes in such a way that FORTH executes an ABORT, OPOP will never be reached and a stray item will be left on the OSTACK. If repeated enough, this could overflow the stack.

To solve the problem, you need to modify ABORT to reset OSTACK. The way to do this depends on your FORTH system.

This is the actual code that you'll need:

```
: ORESET    OSTACK DUP 16 + SWAP ! ; ( Reset
OSTACK to empty)
: MYABORT    ORESET ABORT ;
```

To replace ABORT by MYABORT, a variety of routes are open. In a good professional FORTH system, ABORT will be vectored through a variable, or other known memory location, to let you set up custom error handling. The variable might be called, for example, UABORT, and you would use:

```
' MYABORT UABORT !
```

You need to find out how it's done in your own system.

If your system does not provide vectored execution for ABORT, then you need to check out the internals of the definition of your ABORT, preferably by using a decompiler, but at worst by studying its hex dump.

At some point, usually at the very end, ABORT calls QUIT. If you can find the address of this call to QUIT (let's call it ZZZZ), then instead of MYABORT you can define:

```
: MYQUIT    ORESET QUIT ;
and then do a dirty patch of MYQUIT into this
address:
' MYQUIT ZZZZ !
```

Test the redefined ABORT carefully before proceeding, as a flaky ABORT is pretty explosive in effect.

types.doc

"Object-Oriented FORTH," by Dick Pountain, August, page 227. Also see readfort.me.

Screen # 1

```
0 ( Type definitions
1
2 ( Working variables for object compiler)
3 VARIABLE SIZE      ( Holds storage size of type)
```

(continued)

```

4 VARIABLE INHERIT      ( Holds address of inherited
                        ops vocab)
5 VARIABLE OPS          ( Holds address of end of
                        ops vocabulary)
6 VARIABLE STASH        ( Temporary store for current
                        vocabulary)
7 VARIABLE PUBLIC       ( Holds link to ordinary
                        dictionary)
8 VARIABLE LASTLOCAL    ( Holds address of last word
                        in type)
9 VARIABLE IN.TYPE.DEF? ( Flag; are we in a type
                        definition?)
10 VARIABLE VAL         ( Flag; has a VAL index
                        been declared?)
11
12 0 CONSTANT FALSE     FALSE NOT    CONSTANT TRUE
  ( For readability)
13
14 ( I refuse to use THEN, which is a syntactic
  abomination!!)
15 : ENDIF [COMPILE] THEN ; IMMEDIATE    -->

```

Screen # 2

```

0 ( Type definitions
1
2 ( Make a third stack to hold current object's
  address ; its size
3  determines how deeply type definitions
  may be compounded)
4  CREATE OSTACK HERE 16 + , 16 ALLOT
5
6 ( Push parameter stack to object stack)
7 : OPUSH OSTACK -2 OVER +! @ ! ; ( n --- )
8
9 ( Pop object stack and discard)
10 : OPOP 2 OSTACK +! ; ( --- )
11
12 ( Copy top of object stack and ADD to top
  of parameter stack)
13 : OCOP+ OSTACK @@ + ; ( n --- n )
14
15                                     -->

```

Screen # 3

```

0 ( Type definitions
1
2 ( Compile offset into instance variable and
  bump the total)
3 : OFFSET SIZE @ 2 + , SIZE +! ; ( size --- )
4
5 ( Purely for brevity)
6 : COMPLIT [COMPILE] LITERAL ;
7
8 ( Compile code to add offset into object body)
9 : COMPILE.ADDOFF COMPLIT COMPILE OCOP+ ;
10
11 ( Create a new instance variable of 'size' bytes)
12 : VAR CREATE OFFSET ( size --- )
13      IMMEDIATE
14      DOES> @ COMPILE.ADDOFF ;
15                                     -->

```

Screen # 4

```

0 ( Type definitions
1
2 ( Open a type declaration)
3 : TYPE> LATEST PUBLIC ! ( NFA of last public word)
4      CREATE ( Make a header)
5      HERE LASTLOCAL ! ( Store its PFA)
6      0 SIZE ! ( Initialisations)
7      TRUE IN.TYPE.DEF? !
8      FALSE INHERIT ! ;
9
10 ( Mark boundary which hides the
  instance variables)
11 : OPS> HERE ( Address following
              last VAR)

```



```

12      0 C,          ( Make dummy name field)
13      LATEST ,      ( Link field points to
                      last VAR)
14      DUP CONTEXT @ ! ( Let Forth know about
                      dummy word)
15      N>LINK OPS ! ; ( Save its LFA)  -->

```

Screen # 5

```

0 ( Type definitions
1
2 ( Save current vocabulary; set operations
  vocabulary)
3 : UNLOCK CONTEXT @ DUP @ STASH ! ! ;
  ( key --- )
4
5 ( Restore current vocabulary)
6 : LOCK STASH @ CONTEXT @ ! ;
7
8 ( Look up an operation in its type vocabulary)
  ( key --- CFA)
9 : FINDOP BL WORD SWAP ( Get operation name)
10 UNLOCK FIND LOCK ( Find it)
11 0= ABORT" unrecognised operator " ;
  ( Abort if not found)
12
13
14
15 -->

```

Screen # 6

```

0 ( Type definitions
1
2 ( Execute an operation immediately, if found)
3 : DO.OP SWAP OPUSH FINDOP EXECUTE OPOP ;
  ( addr key --- ?)
4
5 ( Compile operation calling sequence)
6 : COMPILE.CALL COMPILE OPUSH , COMPILE OPOP ;
  ( CFA --- )
7
8 ( Look-up operation and compile it)
9 : COMPILE.OP FINDOP SWAP COMPLIT
  ( addr key --- )
10 COMPILE.CALL ;
11
12 ( Fetch size field contents from instance
  variable or type)
13 : SZ@ 2 + @ ; ( addr --- size)
14
15 : SELF ; ( Optional; used for readability only) -->

```

Screen # 7

```

0 ( Type definitions
1
2 ( Create an instance variable of a predefined type)
  ( addr --- )
3 : MAKE.STRUCTVAR DUP SZ@ ( get size)
4 SWAP @ ( Get key)
5 CREATE , OFFSET ( Store key and size)
6 IMMEDIATE
7 DOES> DUP @ SWAP SZ@ 2 - ( Get key and offset)
8 COMPILE.ADDOFF ( Compile code.... )
9 FINDOP ( to treat as.... )
10 COMPILE.CALL ; ( an object. )
11
12 ( Compile or interpret an operation
  according to state)
13 : DO.OR.COMP STATE @ IF COMPILE.OP
  ( addr key --- )
14 ELSE DO.OP
15 ENDIF ; -->

```

Screen # 8

```

0 ( Type definitions
1

```

(continued)

```

2 ( Allot space initialised to zeroes)
3 : ALLOTZ  DUP HERE SWAP 0 FILL ALLOT ; ( n --- )
4
5 ( Execute an operation called INIT if
  there is one)
6 : INITIALIZE SWAP OPUSH
7   UNLOCK LIT" INIT" FIND LOCK
8   IF EXECUTE ELSE DROP ENDIF OPOP ;
9
10 ( Create a new instance of a type)
   ( addr --- )
11 : MAKE.INSTANCE CREATE HERE SWAP
12   DUP @ DUP ,      ( Store key into instc)
13   SWAP SZ@ ALLOTZ ( Allot its storage)
14   INITIALIZE      IMMEDIATE
15   DOES> DUP @ DO.OR.COMP ;      -->

```

Screen # 9

```

0 ( Type definitions
1
2 : INCLUDE> ' >BODY @ INHERIT ! ;
  ( Inherit ops from old type)
3
4 ( Juggle dictionary pointers to seal the type body)
5 : LINKS HERE BODY> >LINK PUBLIC @ SWAP ! ( --- )
6   LASTLOCAL @ BODY> >NAME OPS @ !
7   INHERIT @ LASTLOCAL @ BODY> >LINK ! ;
8
9 ( Close type declaration)
10 : ENDTYPE> LATEST CREATE LINKS
   ( Close the body)
11   , SIZE @ ,      ( Store key and size)
12   FALSE IN.TYPE.DEF? !
13   DOES> IN.TYPE.DEF? @ IF MAKE.STRUCTVAR
14   ELSE MAKE.INSTANCE
15   ENDIF ;      -->

```

Screen # 10

```

0 ( Array definitions
1
2 ( Calculate address of array element)
3 : INDEX+ ROT * + ; ( index PFA width --- addr)
4
5 ( Interpret an array operation)
   ( index PFA key --- )
6 : ARRAY.DO.OP FINDOP
   ( Get operation CFA)
7   ROT ROT 4 + DUP @
   ( Get width of element)
8   INDEX+
   ( Calculate element address)
9   OPUSH EXECUTE OPOP ; ( Do it!)
10
11 ( Place index on stack at compile time)
12 : VAL[ TRUE VAL ! [COMPILE] [ ; IMMEDIATE
13
14 ( Reset the VAL flag)
15 : ~VAL FALSE VAL ! ;      -->

```

Screen # 11

```

0 ( Array definitions
1
2 ( Compile an array operation)
   ( {index} PFA key --- )
3 : ARRAY.COMP.OP FINDOP >R
   ( Get op CFA and stash it)
4   4 + DUP @
   ( Get width of array)
5   VAL @
   ( Index at compile time?)
6   IF INDEX+ COMPLIT
   ( Compile el. addr)
7   ELSE SWAP COMPLIT COMPLIT
   ( or code to calc )
8   COMPILE INDEX+ ( at runtime)
9   ENDIF

```



```

10      R> COMPILE.CALL ~VAL ;
      ( compile op call)
11
12 ( Compile or interpret an array op)
  ( index PFA key --- )
13 : ARRAY.DO.OR.COMP STATE @ IF ARRAY.COMP.OP
14      ELSE ARRAY.DO.OP
15      ENDIF ;      -->

```

Screen # 12

```

0 ( Array definitions
1 ( Create a typed array as an instance variable)
  ( count PFA --- )
2 : ARRAY.VAR CREATE DUP @ , OVER ,
  ( Store key and count)
3     SZ@ DUP ,      ( Store width of element)
4     *              ( Size = count * width)
5     OFFSET        ( Store offset etc.)
6     IMMEDIATE
7     DOES> DUP @      ( Get key)
8     FINDOP >R      ( Get op CFA and stash it)
9     DUP 6 + @      ( Get offset)
10    2 - SWAP 4 + @  ( Get width)
11 ( Compile el addr) VAL @ IF INDEX+ COMPILE.ADDOFF
12 ( or code to...) ELSE SWAP COMPLIT COMPLIT
13 ( calculate it...) COMPILE INDEX+
14 ( at runtime) COMPILE OCOP+
15 ENDIF R> COMPILE.CALL ~VAL ;      -->

```

Screen # 13

```

0 ( Array definitions
1
2 ( Make a new array instance) ( count PFA --- )
3 : MAKE.ARRAY CREATE 2DUP @ , ,
  ( Store key and count)
4     SZ@ DUP , SWAP
  ( Store width)
5     * ALLLOTZ
  ( Allot the space)
6     IMMEDIATE
7     DOES> DUP @ ARRAY.DO.OR.COMP ;
8
9 ( Create an array object or variable)
  ( count +++ )
10 : ARRAY-OF ' >BODY
11     IN.TYPE.DEF? @
12     IF ARRAY.VAR
13     ELSE MAKE.ARRAY
14     ENDIF ;
15

```

Screen # 14

```

0 ( DATA STRUCTURES EMPLOYED INTERNALLY
1
2 OBJECT <----- width ----->
3 +-----+
4 | header | key | storage fields |
5 +-----+
6
7 ARRAY-OF OBJECTS
8 +-----+
9 | header | key | count | width | elements |
10 +-----+
11
12 TYPE DEFINING WORD
13 +-----+
14 | header | key | size |
15 +-----+ )

```

Screen # 15

```

0 (
1 VAR NAME
2 +-----+
3 | header | offset |
4 +-----+
5

```

(continued)

```

6 STRUCTVAR NAME
7  +-----+-----+
8  | header | key  | offset |
9  +-----+-----+
10
11 ARRAY.VAR NAME
12  +-----+-----+-----+-----+
13  | header | key  | count | width | offset |
14  +-----+-----+-----+-----+
15                                     )

```


September

breakpt.asm

Programming Insight: "Breaking Out," by Edward Batutis. September, page 127. Also see brkptcom.bas.

```
; BREAKPT
; Copyright 1985, Edward Batutis
;
; Invokes the breakpoint interrupt when Ctrl-Shift-Shift
; combination is pressed.

; Assembled with the IBM Macro Assembler Version 1.00

cseg    segment para    public 'code'
        assume cs:cseg,ds:cseg
        org    100h

breakpt proc

        jmp    install

copyright    db    'BREAKPT (c) Copyright 1985,'
            db    ' Edward J. Batutis',01ah

old_int9_vector label    dword
old_int9_offs    dw    ?
old_int9_seg    dw    ?

new_int9:

    ; call old keyboard routine by simulating an int

    pushf
    call    cs:old_int9_vector

    push    es                ; save registers
    push    ax
    push    bx

    mov     ax,40h            ; look at keyboard flag1 in
    mov     es,ax            ; ROM BIOS data area
    mov     bx,17h
    mov     al,es:[bx]
    and     al,07h           ; mask off everything but lowest
                                ; three bits
    cmp     al,7              ; are Ctrl-Shift-Shift depressed?
    jne     quit              ; no, quit

    ; turn on the trap flag

    pop     bx                ; restore registers
    pop     ax
    pop     es

    push    ax                ; save register
    pushf    ; get flags into ax
    pop     ax

    or      ax,0100h          ; set trap flag on
    push    ax                ; put new flags back
    popf
    pop     ax                ; restore ax
    nop                    ; wait one instruction
    iret                    ; debug is invoked at this instruction
```

(continued)

```

quit:      pop     bx           ; restore registers when
           pop     ax           ; quitting
           pop     es
done:      iret

END_OF_RESIDENT_CODE LABEL BYTE

banner db    'BREAKPT installed.',10,13,'$'

install:

           ; get interrupt vector for
           ; keyboard interrupt

           mov     ah,35h       ; get interrupt vector function
           mov     al,9         ; get interrupt 9
           int     21h
           cmp     bx,offset new_int9 ; are we already installed?
           je      no_install   ; yes, just exit

           mov     old_int9_offs,bx ; save old keyboard interrupt
           mov     old_int9_seg,es ; address

           mov     dx,offset banner ; print banner
           mov     ah,9         ; print string function
           int     21h

           ; set keyboard interrupt
           ; to point to new_int9
           mov     ah,25h       ; set interrupt function
           mov     al,9         ; set interrupt 9
           mov     dx,offset new_int9 ; point to new routine
           int     21h

           ; terminate, but stay
           ; partially resident
           ; point to last byte of resident
           ; routines+1

           mov     dx,offset END_OF_RESIDENT_CODE+1
           int     27h

no_install:
           int     20h          ; don't install, just exit

breakpt endp

cseg      ends
end        breakpt

```

brkptcom.bas

Programming Insight: "Breaking Out," by Edward
Batutis. September, page 127. Also see breakpt.asm.

```

100 'RUN THIS PROGRAM TO CREATE breakpt.com
110 PRINT "Creating breakpt.com"
120 OUTFILE$="breakpt.com"
140 DATA &hEB, &h74, &h90, &h42, &h52, &h45, &h41, &h4B, &h50, &h54
150 DATA &h20, &h28, &h63, &h29, &h20, &h43, &h6F, &h70, &h79, &h72
160 DATA &h69, &h67, &h68, &h74, &h20, &h31, &h39, &h38, &h35, &h2C
170 DATA &h20, &h45, &h64, &h77, &h61, &h72, &h64, &h20, &h4A, &h2E
180 DATA &h20, &h42, &h61, &h74, &h75, &h74, &h69, &h73, &h1A, &h00
190 DATA &h00, &h00, &h00, &h9C, &h2E, &hFF, &h1E, &h31, &h01, &h06
200 DATA &h50, &h53, &hB8, &h40, &h00, &h8E, &hC0, &hBB, &h17, &h00
210 DATA &h26, &h8A, &h07, &h24, &h07, &h3C, &h07, &h75, &h0E, &h5B
220 DATA &h58, &h07, &h50, &h9C, &h58, &h0D, &h00, &h01, &h50, &h9D
230 DATA &h58, &h90, &hCF, &h5B, &h58, &h07, &hCF, &h42, &h52, &h45
240 DATA &h41, &h4B, &h50, &h54, &h20, &h69, &h6E, &h73, &h74, &h61
250 DATA &h6C, &h6C, &h65, &h64, &h2E, &h0A, &h0D, &h24, &hB4, &h35
260 DATA &hB0, &h09, &hCD, &h21, &h81, &hFB, &h35, &h01, &h74, &h1D
270 DATA &h89, &h1E, &h31, &h01, &h8C, &h06, &h33, &h01, &hBA, &h61
280 DATA &h01, &hB4, &h09, &hCD, &h21, &hB4, &h25, &hB0, &h09, &hBA

```



```

290 DATA &h35, &h01, &hCD, &h21, &hBA, &h62, &h01, &hCD, &h27, &hCD
300 DATA &h20
310 TOTAL= 161
320 OPEN OUTFILE$ AS #1 LEN=1
330 FIELD #1,1 AS A$
340 FOR I=1 TO TOTAL
350     READ A
360     LSET A$=CHR$(A)
370     PUT 1
380 NEXT
390 CLOSE
400 PRINT"Done."
410 END

```

ccitt.c

Programming Project: "Calculating CRCs by Bits and Bytes,"
by Greg Morse. September, page 114. Also see xmodem.c,
sdlc.asm, and xmodem.asm.

```

/* Straightforward, non-optimized CRC-CCITT
   routine */
/* Assumes 16-bit integer variables */
/* MSB of integer is MSB of CRC result */
#define POLY 0x8408
/* POLY = 1021 in bit rev order */
BLKCRC(bufptr, crcres, count)
    unsigned char *bufptr;
    unsigned int *crcres, count;
{
    int i;
    *crcres = 0; /* for SDLC use 0xFFFF */
    for (i=1; i<=count; ++i, bufptr++) /* do for whole BLK */
        bytecrc(bufptr, crcres) /* do CRC for 1 char */
    return (*crcres);
} /* end BLKCRC */

bytecrc(bufptr, crcres)
    unsigned char *bufptr;
    unsigned int *crcres;
{
    unsigned int j,ch,Q;
    ch = (unsigned int) *bufptr; /* get char, to int fmt */
    for (j=1; j<=8; j++) { /* do each bit LSB 1st */
        Q=(*crcres&0x0001)^(ch&0x0001)/* Q=R0 XOR D */
        if (Q == 0x0001) { /* Q is one */
            *crcres= *crcres>>1; /* shift right one */
            *crcres= *crcres^POLY; /* XOR with number */
        }
        else /* Q is zero */
            *crcres= *crcres>>1; /* just shift no XOR */
        ch = ch >>1; /* move next data bit */
        /* into position */
    } /* end FOR - data bits all done */
    return (*crcres);
} /* end bytecrc */

```

exps.st1

"Atari ST Software Development," by Michael Rothman.
September, page 223.

***** Example 1 *****

```

/* format.c */
/* Format a floppy disk on the ST. Return 0 if sucess,*/
/* else non-zero.
Edit History:
000 19-Sep-85 MR Creation
001 07-Feb-86 MR Modified for BYTE article

```

(continued)

```

*/
/* #define DEBUG 1 */
#include <osbind.          /* C bindings for OS routines */

#define HITRACK 79         /* Highest numbered track */
#define SECTORS 9          /* Sectors per track */
#define MAGIC 0x87654321L /* Required by Flopfmt() */
#define VIRGIN 0xE5E5      /* Pattern to write to sectors */
#define ILEAVE 1           /* Interleave factor */
#define DISKTYPE 2         /* Single side, 80 track */
#define NOLOAD 0           /* No loader code in boot sector */
#define RANDOM 0x1000000L /* make protobt make a random */
#define BOOTSECT 1        /* Side 0, sect 1 gets boot sect */
#define TRACK0 0           /* Track for boot sector */
#define SIDE0 0            /* Format side 0 */

extern void errprint();
/* error notification routine */

format(devno)
int devno; /* device holding media to format */
{
/* Automatic variables */
/* count tracks */
register int i;
/* buffer for track, protoboot */
register char *buf;
/* success in format? */
register int succ, totsucc = 0;
/* doesn't do anything */
long filler;

/* Code */
/* Allocate memory for track. The ST formats one
track at a time, and requires sufficient RAM to
verify that track in memory. Malloc is a GEMDOS
call. */
buf = Malloc(8192L);
if (buf == 0L)
{
#ifdef DEBUG
errprint(0, "insuff memory for format");
#endif
return(-1);
}

/* Format each track. VIRGIN is the value to write
to the newly formatted track. This particular value
(0xE5E5) is suggested in the documentation, but many
values are possible. Flopfmt is XBIOS. */
for (i=HITRACK; i>=0; i--)
{
succ = Flopfmt(buf, filler, devno, SECTORS,
i, SIDE0, ILEAVE, MAGIC, VIRGIN);
totsucc += succ;
}

/* Release memory. GEMDOS */
Mfree(buf);

/* For the purposes of this routine, I won't accept
any bad sectors. But, if there were any, their numbers
would have been left in the buffer, buf, after each
track was formatted. I could alternatively retried,
or recorded the bad sectors if I were developing my
own file system. */

if (totsucc != 0)
{
#ifdef DEBUG
errprint(totsucc, "format failed");
#endif
return(-1);
}
}

```



```

/* Now we need to put a boot sector on the disk. */
/* Allocate a 512 byte buffer */
buf = Malloc(512L);
/* Prototype a boot sector in that buffer.
The second parameter is a serial number for the
disk. The value I have chosen asks the XBIOS
to generate a random number. XBIOS */
Protobt(buf,RANDOM,DISKTYPE,NOLOAD);
/* Write out the boot buffer to track 0, side 0.
Last parameter is how many sectors to write. */
succ = Flopwr(buf,filler,SIDENO,BOOTSECT,TRACK0,SIDE0,1);
/* Throw away memory */
Mfree(buf);

/* Return success or failure */
return (succ);
}

```

***** Example 2 *****

This is a trap handler of the sort you might use if you were programming the ST in 68000 assembler, or you were using a high level language and needed to write a binding for access to a BIOS, XBIOS or GEMDOS function. The functions assume the C calling conventions, that is, if there are any parameters, they are assumed to have been pushed onto the stack in reverse order, and to be no smaller than a word (16 bits). The number of the routine itself must be pushed last, just before the trap call. Of course if you are developing using the C provided in Atari's kit for developers, this process will be transparent to you, since a set of bindings is available which makes TOS calls look just like ordinary C function calls.

```

; At entry, any arguments for the function have been
; pushed on the stack
; in reverse order, C-style. Then the function number
; was pushed.
; Finally this routine was called, so as we enter the
; return address of
; the caller is on top of the stack.

```

```

retsv: ds.l ; some memory for a long variable

```

```

traprtn:
move.l (a7)+, retsv ; Save off the return address,
                    ; because the OS functions don't
                    ; expect it.
trap #13 ; Trap to the BIOS function. (BIOS
          ; is available through trap 13,
          ; XBIOS through 14, GEMDOS through
          ; trap 1).
move.l retsv, -(a7) ; Put the return address back on
                    ; stack.
rts ; Return to caller.

```

***** Example 3 *****

```

/* SAMPLE.C Original version provided by Atari as
part of the developer's kit. Rearranged, cleaned up,
defines added, many variable names changed, and all
comments added by M.R.
*/

/* This program is a simple example of use of some

```

(continued)

VDI primitives and certain parts of the AES, in particular the window library and the event handler. It opens a window on the desktop and draws a filled ellipse in it. It waits for the user to move the window or resize it, and then redraws the ellipse. If the user selects the window close box, the program closes the window and then terminates.

```
*/
```

```
/* Defines. These are all just for readability. */
```

```
#define COLOR0 0
#define COLOR1 1
#define SCREEN 1
#define SOLID 1
#define PATTERN 2
#define USER 4
#define DOT 1
#define SYSTEM 1
#define RC 2
#define ARROW 0
#define WORKAREA 4
#define WNAME 2
#define WINDAREA 1
#define CLIPON 1
#define WM_REDRAW 20
#define WM_CLOSED 22
#define WM_SIZED 27
#define WM_MOVED 28
#define WF_CURRXYWH 5
```

```
/* Window type bits */
```

```
#define NAME 0x0001
#define CLOSE 0x0002
#define MOVE 0x0008
#define SIZE 0x0020
```

```
/* These arrays are used by the VDI in its own code.
The developer is expected to allocate room for them
somewhere in the application. */
```

```
int contrl[12], intin[256], ptsin[256], intout[256],
ptsout[256];
```

```
main()
{
```

```
    /* Local variables */
```

```
/* For the workstation */
```

```
int workin[10]; /* Input values for v_openvwk() */
int workout[56]; /* Output values for v_openvwk() */
int shandle; /* Workstation (screen) handle */
```

```
/* Variables for our application's window */
```

```
int whandle; /* Window handle */
int wind_type; /* Holds window attribute bits */
int xwork; /* X coordinate, upper left hand corner,
work area of window */
int ywork; /* Y coordinate, upper left hand corner,
work area of window */
int wwork; /* Width, work area of window */
int hwork; /* Height, work area of window */
int xbord; /* X coordinate, upper left hand corner,
border of window */
int ybord; /* Y coordinate, upper left hand corner,
border of window */
int wbord; /* Width, border area of window */
int hbord; /* Height, border area of window */
int xcen, ycen; /* Coordinates of central point in window */
```

```
/* These four variables are returned by graf_handle()
(described later). They are not used further in
this application. */
```

```
int gr_wchar, gr_hchar; /* Width and height of a character
```



```

                                cell for font used in menus and
                                dialogs */
int gr_wbox, gr_hbox; /* Width and height of box large
                        enough to hold a system font
                        character */

/* Miscellaneous */
int ap_id; /* Application id */
int clip[4]; /* Holds coordinates defining clip region */
int mgbuf[8]; /* Buffer for message events from AES */
int dummy; /* Word buffer for miscellaneous use */

/* Begin program */

/* Appl_init is the AES initialization routine.
It makes the AES aware of the application, and
initializes certain AES data structures. The
application id it returns can be used later by
the application for other AES routines. */

ap_id = appl_init();

/* When the application starts, GEM has already opened
the screen workstation for its own use. Therefore the
screen has a workstation handle (identifier). We need
this handle to open our own virtual workstation with the
attributes we would like. The routine graf_handle
(part of the AES graphics library) returns this handle.
It also puts certain character size info into the passed
parameters (which we don't happen to need). */

shandle = graf_handle(&gr_wchar, &gr_hchar, &gr_wbox, &gr_hbox);

/* Now we can open a virtual workstation. If the
attributes of the GEM opened physical workstation were
to our liking, we could just use it. Or we could change
the attributes with VDI attribute calls. As often is
the case, GEM provides redundancy - a decision as to
elegance is up to you.

The workin array will define the default attributes we
want. Note these are all defaults. We will modify some
during the main loop of the program. */

/* The device type - 1 is for screen */
workin[0] = SCREEN;

/* Set the line type to a solid line */
workin[1] = SOLID;

/* Polyline color index to 1 (The background color
defaults to whatever is in color register 0.
The foreground to whatever is in register 1. So by
selecting 1, we are selecting the default foreground
color. Incidentally, on a monochrome system, 0
defaults to white, and 1 to black. Thus the screen
has that black on white, Mac-like appearance). */
workin[2] = COLOR1;

/* Marker type 1 - which happens to be a dot.
This is set for completeness; there is no use of the
polymarker routine in this application. */
workin[3] = DOT;

/* Polymarker color set to 1. Ditto */
workin[4] = COLOR1;

/* Text face to the system face */
workin[5] = SYSTEM;

/* Text color */
workin[6] = COLOR1;

/* Fill interior style. The available styles for a
fill are hollow, solid, pattern, hatch, and user-defined.
Hollow fills with the background color (index 0).

```

(continued)

Solid with the currently selected fill color. The other three choices are further modified by the fill style index (see the next entry in array) to pick the fill pattern.

```
*/
workin[7] = SOLID;

/* Fill index style. This modifies fill interior style
to select the actual fill style (unless the interior
style is solid or hollow). We'll just select the
default 1. */
workin[8] = 1;

/* Fill color. */
workin[9] = COLOR1;

/* The last entry in the array selects either raster
coordinates (machine specific) or normalized device
coordinates (portable). */
workin[10] = RC;
```

```
/* Open our virtual workstation, passing it the
input array, the physical workstation handle, and
the output array. Note, the virtual workstation's
handle is returned in the same variable which held
the physical handle. A VDI routine. */
```

```
v_opnvwk(workin, &shandle, workout);
```

```
/* Set the mouse cursor form to the arrow. Yes,
this is the default anyway, but this is supposed
to be an example program, you know? The second
parameter in graf_mouse is a dummy in this case. If
the mouse form selected was not pre-defined, the
second parameter would have to be a pointer to
a mouse form definition block - a data structure
for defining a mouse-driven cursor. Graf_mouse is
an AES routine. */
```

```
graf_mouse(ARROW,&dummy);
```

```
* Time to get into the meat of the problem. We want
to create a window for our application. The first thing
to do is get the size of the desktop window. This is
a window GEM creates when the system boots. Since it
occupies the entire screen, it is a guide to the maximum
area our application can use. The function wind_get can
be used to get various values; here we are going to
get the size of the work area (area not including border)
of the desktop window (the first parameter is 0, indicating
the desktop window). Since we intend OUR window to use
this entire area, we put the results into the variables
we will be using to define the border of our window. This
and all the functions beginning wind_ are AES functions. */
```

```
wind_get(0,WORKAREA,&xbord,&ybord,&wbord,&hbord);
```

```
/* Now we know the size for our window. To create it,
we need to specify its attributes, and its border size. */
```

```
/* We want a window with a name, a close box, a move
bar and a size box... */
```

```
wind_type = NAME | CLOSE | MOVE | SIZE;
```

```
/* O.K. Make me a window. This call sets up
internal data structures and establishes the maximum
size for the window, but it does not actually draw
the window on the screen. */
```

```
whandle = wind_create(wind_type,xbord,ybord,wbord,hbord);
```

```
/* We specified we wanted a window with a name,
but now we need to specify what that name is.
This function can also be used to set or change
other window parameters. */
```



```

wind_set(whandle,WNAME,"SAMPLE",0,0);

/* At last, draw the window. We have decided to draw
it initially in its full size, but we could vary the
last four parameters of this function if we desired
otherwise. */

wind_open(whandle,xbord,ybord,wbord,hbord);

/* Incidentally, we have never found out the size of
the work area of our window. We'll need that later,
so let's do it with the wind_calc function. This
function, given the window type can determine either
the border or the work area dimensions. */

wind_calc(WINDAREA,wind_type,xbord,ybord,wbord,hbord,
          &xwork,&ywork,&wwork,&hwork);

/* Now, draw the filled ellipse in our window and
wait for messages from the AES. If it's a window
resize or move message, recalculate the window work
area, clear it, and redraw the ellipse. */

do
{
/* If the user expands the size of the window, not only will
the AES send a resize message, it will also send a redraw
message, on the assumption that more of the window is
visible, and you might want to update it. We are redrawing
the entire visible portion of the window work area on receipt
of a resize message anyway, so we can ignore the redraw
message. */

if (mgbuf[0] != WM_REDRAW)
{
/* Hide the cursor. Otherwise area under it will not
be affected by VDI routines. AES. */
v_hide_c(shandle);

/* Set up the clipping rectangle to equal the work
area of the window. Actually, we have no intention
of drawing outside these boundaries anyway, but as
I said, this is a demo program, so here's how you
specify clip. All subsequent graphic primitives in
the VDI will respect these boundaries, unless the
routine is called again with a different clip
rectangle, or with the second parameter set to 0
(which means turn clipping off altogether). VDI. */

clip[0]=xwork;
clip[1]=ywork;
clip[2]=xwork+wwork-1;
clip[3]=ywork+hwork-1;
vs_clip(shandle,CLIPON,clip);

/* Let's clear the window work area to background color.
There are lots of ways to do this. Here we set the
interior fill style to pattern, the interior style index
to 8 (which selects a "solid" pattern), and the fill
color to 0, the background color. Then we call a
primitive - v_bar, which draws a filled bar of the
size defined by its second parameter (for which we
used the already available clip rectangle). All these
routines are VDI. */

vsf_interior(shandle,PATTERN);
vsf_style(shandle,8);
vsf_color(shandle,COLOR0);
v_bar(shandle,clip);

/* Set the fill interior and color
to the desired style. Calculate the ellipse
center point. And voila! - draw the ellipse.
These are all VDI routines. */

```

(continued)

```

vsf_interior(shandle,USER);
vsf_color(shandle,1);
xcen=xwork+wwork/2;
ycen=ywork+hwork/2;
v_ellipse(shandle,xcen,ycen,wwork/2,hwork/2);

/* Reshow the cursor, since we're done drawing. AES. */
v_show_c(shandle);
}
/* End of the part we don't do on a redraw message. */

/* O.K. Go to sleep until we get a message that the user
has done something interesting. AES. */
evnt_mesag(&mgbuf);

/* If user wanted to resize window or move it, the
new border coordinates will have been left in the message
buffer, positions 4 through 7. Use them to recalculate
the work area size and reset the coordinates of the entire
window. */

if (mgbuf[0] == WM_SIZED || mgbuf[0] == WM_MOVED)
{
wind_calc(WINDAREA,wind_type,mgbuf[4],mgbuf[5],
          mgbuf[6],mgbuf[7],&xwork,&ywork,
          &wwork,&hwork);
wind_set(whandle,WF_CURRXYWH,mgbuf[4],mgbuf[5],
          mgbuf[6],mgbuf[7]);
}
/* Repeat until the message from the AES says
the user clicked the window close box. */
while (mgbuf[0] != WM_CLOSED);

/* Close and delete the window. The data structure
remains allocated until a window delete. */

wind_close(whandle);
wind_delete(whandle);

/* Close the virtual workstation. */
v_clsvwk(shandle);

/* Tell the AES the application is done,
and terminate */
appl_exit();
}

```

sdlc.asm

Programming Project: "Calculating CRCs by Bits and Bytes,"
by Greg Morse. September, page 114. Also see xmodem.c,
ccitt.c, and xmodem.asm.

```

* part A: calling main pgm for SDLC subroutine
*
NAM    SDLC
Stt1   Calculate SDLC FCS a byte at a time
Ifp1   use /d0/defs/os9defs
use /d0/defs/os9defs
endc
mod CRCsiz,CRCnam,prgrm+objct,rcnt+1,CRCBeg,CRCMem
CRCNam  fcs "SDLC"
        org  0
CRCReg  equ  . ;;keep
CRCHI   rmb  1 ;;      these
CRCLO   rmb  1 ;;      statements
Temp    rmb  1 ;;      together
stakbot rmb  200-.
CRCMem  equ  .
*
CRCBeg  equ  *

```



```

lslo      A= 0 T3 T2 T1 T0 0 0 0
lslo      A= T3 T2 T1 T0 0 0 0      (28)
eora Temp A= U7 U6 U5 U4 0 0 0      (32)
eora CRCLo A= U7 U6 U5 U4 T3 T2 T1 T0 (36)
sta Temp  temp= U7 U6 U5 U4 T3 T2 T1 T0 (40)
clrb      (42)
lsra      A= 0 U7 U6 U5 U4 T3 T2 T1
rorb      B= T0 0 0 0 0 0 0 0      (46)
eorb Temp B= V7 U6 U5 U4 T3 T2 T1 T0 (50)
lsra      A= 0 0 U7 U6 U5 U4 T3 T2
rorb      B= T1 V7 U6 U5 U4 T3 T2 T1 (54)
lsra      A= 0 0 0 U7 U6 U5 U4 T3
rorb      B= T2 T1 V7 U6 U5 U4 T3 T2 (58)
lsra      A= 0 0 0 0 U7 U6 U5 U4
rorb      B= T3 T2 T1 V7 U6 U5 U4 T3 (62)
lsra      A= 0 0 0 0 0 U7 U6 U5
rorb      B= U4 T3 T2 T1 V7 U6 U5 U4 (66)
eorb CRCHI B= new CRCLo      (70)
eora Temp A= new CRCHI      (74)
std CRCReg one data byte all done (79)
RC.99 equ  *make RTS if doing only 1 byte
cmpx ,S    x past end of buffer?   (83)
blo CRC.10 if not repeat inner loop Note: (86)
           B still = CRCLo
leas 2,s   pop topaddr
puls y,pc  restore y and return
*FoxMsg equ *
fcc /THE,QUICK,BROWN,FOX,0123456789/
foxsiz equ  *-foxmsg
foxcrc fdb 0 room for fox FCS bytes
USE shoregs.src utility subrtn to print regs
emod      os9 directive
CRCSiz equ *

```

skam.bas

"Keyed File Access in BASIC," by Stephen C. Perry.
September, page 137. Also see skam1.bas.

```

1  '-----
2  '      SAMPLE PROGRAM USING KEYED ACCESS ROUTINES      -
3  '-----
5  UA$="A"      ' .. DRIVE CONTAINING DATA
16 OPEN "R",#2,UA$+":DATA.EMP",84      ' .. OPEN DATA FILE
17 FIELD #2, 9 AS KY$, 20 AS NM$, 6 AS BD$, 1 AS SX$, 3 AS JC$, 20 AS A1$, 20
AS A2$, 5 AS ZP$
18 '
19 '      KY$ - ZIP CODE (KEY) JC$ - JOB CODE
20 '      NM$ - NAME          A1$ - STREET ADDR.
21 '      BD$ - BIRTH DATE   A2$ - CITY-STATE
22 '      SX$ - SEX          ZP$ - ZIP CODE
23 '
25 MX%=150: F1$="PTR.EMP"      ' ..INDEX FILE NAME
30 II%=1: GOSUB 2000      ' ..INITIALIZE DATA STRUCTURE
31 '
32 INPUT "OPERATION (D,A,L,S,LA,U,Q)";Q$
33 IF Q$="D" THEN GOSUB 150: GOTO 32      ' DELETE
34 IF Q$="L" THEN GOSUB 180: GOTO 32      ' LIST INDIVIDUAL DATA
35 IF Q$="A" THEN GOSUB 100: GOTO 32      ' ADD
36 IF Q$="S" THEN II%=8: GOSUB 2000: GOTO 32      ' DISPLAY STATISTICS
37 IF Q$="LA" THEN GOSUB 200: GOTO 32      ' LIST ALL RECORDS
38 IF Q$="U" THEN GOSUB 250: GOTO 32      ' UPDATE RECORD
40 IF Q$<>"Q" THEN 32
50 CLOSE: END
97 '
98 ' ***** ADD RECORD
99 '
100 INPUT "SS#";A$: IF A$="END" THEN 120 ELSE IF LEN(A$)<>9 THEN 100
101 II%=5:GOSUB 2000: IF RC%<>0 THEN LSET KY$=A$: GOTO 102 ELSE PRINT "***
ERROR - KEY ALREADY EXISTS": GOTO 100
102 INPUT "NAME";F$: LSET NM$=F$
105 INPUT "BIRTH DATE";F$: LSET BD$=F$
107 INPUT "SEX";F$: LSET SX$=F$

```



```

109 INPUT "JOB CODE";F$: LSET JC$=F$
110 INPUT "STREET";F$: LSET A1$=F$
111 INPUT "CITY-STATE";F$: LSET A2$=F$
112 INPUT "ZIP CODE";F$: LSET ZP$=F$
115 I1%=2: GOSUB 2000      '.. ADD RECORD
116 IF RC%=0 THEN 100 ELSE PRINT "*** ERROR - RECORD CANNOT BE STORED": GOTO
100
120 I1%=7: GOSUB 2000      '.. STORE POINTERS
122 RETURN
147 '
148 ' ***** DELETE RECORD
149 '
150 ST%=0
151 INPUT "CODE TO DELETE";A$: IF A$="END" THEN 156
152 I1%=4: GOSUB 2000
154 IF RC%=0 THEN ST%=1 ELSE PRINT "*** ERROR - KEY DOES NOT EXIST"
155 GOTO 151
156 IF ST%=1 THEN I1%=7: GOSUB 2000      ' RESTORE POINTERS IF RECORD DELETED
158 RETURN
177 '
178 ' ***** LIST INDIVIDUAL RECORD
179 '
180 INPUT "SOCIAL SECURITY NUMBER";A$: IF A$="END" THEN 190
182 I1%=5: GOSUB 2000: IF RC%<>0 THEN PRINT "***ERROR - KEY DOES NOT EXIST":
GOTO 180
183 PRINT " "
184 PRINT "      NAME: ";NM$
185 PRINT "      JOB CODE: ";JC$
186 PRINT "      BIRTH DATE: ";LEFT$(BD$,2);"/";MID$(BD$,3,2);"/";RIGHT$(BD$,2)
187 PRINT "      ADDRESS: ";A1$
188 PRINT TAB(13);A2$:PRINT ""
189 GOTO 180
190 RETURN
197 '
198 ' ***** LIST RANGE OF RECORDS
199 '
200 NX%=0: I1%=6: K%=0
202 NX%=NX%+1: GOSUB 2000
204 IF RC%<>0 THEN 210
205 PRINT KY$,NM$
206 K%=K%+1: IF K%<10 THEN 202 ELSE INPUT ">";Q$      ' .. PAUSE
207 IF Q$<>"END" THEN K%=0: GOTO 202
210 RETURN
247 '
248 ' ***** UPDATE RECORD
249 '
250 INPUT "SS#";A$: IF A$="END" THEN 270
252 I1%=5:GOSUB 2000      '.. FETCH RECORD TO BE UPDATED
254 IF RC%=1 THEN PRINT "*** ERROR - RECORD DOES NOT EXIST":GOTO 250
255 PRINT "NAME: /";NM$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET NM$=F$
257 PRINT "BIRTH DATE: /";BD$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET BD$=F$
258 PRINT "SEX: /";SX$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET SX$=F$
260 PRINT "JOB CODE: /";JC$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET JC$=F$
262 PRINT "STREET: /";A1$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET A1$=F$
263 PRINT "CITY-STATE: /";A2$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET A2$=F$
265 PRINT "ZIP CODE: /";ZP$;"/";: INPUT F$: IF LEN(F$)<>0 THEN LSET ZP$=F$
266 I1%=3: GOSUB 2000      '.. RESTORE UPDATED RECORD
268 PRINT " ": GOTO 250
270 RETURN
1995 '
1996 ' -----
1997 ' - FILE MANAGEMENT SUBROUTINES (I1%,MX%,F1$,A$,PT%,PT$, NX%,RC%) -
1998 ' -----
1999 '
2000 RC%=0: IF I1%<1 OR I1%>8 THEN RC%=1: RETURN
2001 IF I1%=1 THEN 2006: ' ELSE STORE VARIABLES USED BY SUBROUTINES
2004 ZZ%(1)=J$: ZZ%(2)=JJ$: ZZ%(3)=K$: ZZ%(4)=LO$: ZZ%(5)=HI$: ZZ%(6)=Z$
2005 '
2006 ON I1% GOSUB 2035,2080,2090,2100,2150,2200,2250,2280
2007 '
2008 IF I1%=1 THEN 2010: ' ELSE RESTORE VARIABLES USED BY SUBROUTINES
2009 J%=ZZ%(1): JJ%=ZZ%(2): K%=ZZ%(3): LO%=ZZ%(4): HI%=ZZ%(5): Z%=ZZ%(6)
2010 RETURN
2034 REM --- (1) SUBROUTINE (MX%,F1$) --- INPUT POINTERS AND KEYS
2035 IF MX%<1 THEN RC%=1: RETURN
2037 MR%=(INT((MX%+2)/64)+1)*64

```

(continued)


```

2038 DIM PT$(64),PT%(MR%),KE$(MX%),ZZ$(8)
2040 OPEN "R",#1,UA$+"": "+F1$,128 ' INDEX FILE
2042 FOR J%=1 TO 64: FIELD #1,(J%-1)*2 AS DU$, 2 AS PT$(J%): NEXT J%
2050 K%=0: IF LOF(1)=0 THEN 2062
2051 FOR J%=1 TO INT(MR%/64)
2052 GET 1,J% ' .. INPUT RECORD CONTAINING 64 POINTERS
2054 FOR JJ%=1 TO 64: K%=K%+1: PT%(K%)=CVI(PT$(JJ%)): NEXT JJ%
2055 NEXT J%
2056 '
2057 IF PT%(MR%)=0 THEN 2062
2058 FOR J%=1 TO PT%(MR%)+PT%(MR%-1)
2059 GET 2, J%: KE$(J%)=KY$
2060 NEXT J%
2062 RETURN
2079 REM --- (2) SUBROUTINE (MR%,A$, RC%) -- ADD RECORD TO FILE
2080 GOSUB 2500: IF K%>0 THEN RC%=1: GOTO 2088
2083 GOSUB 2520: IF Z%>MR%-1 THEN RC%=2: GOTO 2088
2085 K%=-K%:GOSUB 2540 ' .. INSERT POINTER . PT%(K%)=Z%
2086 KE$(Z%)=A$
2087 PUT 2,Z% ' .. STORE NEW RECORD
2088 RETURN
2089 REM -- (3) SUBROUTINE --- REWRITE RECORD
2090 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2098
2092 PUT 2,PT%(K%) ' .. STORE RECORD
2098 RETURN
2099 REM --- (4) SUBROUTINE (MR%,A$,RC%) --- DELETE A RECORD
2100 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2110
2102 Z%=PT%(K%): IF K%=PT%(MR%) THEN 2107
2104 FOR J%=K% TO PT%(MR%)-1: PT%(J%)=PT%(J%+1): NEXT J%
2107 JJ%=PT%(MR%-1)
2108 PT%(PT%(MR%))=0: PT%(MR%)=PT%(MR%)-1: PT%(MR%-1)=JJ%+1:PT%(MR%-2-JJ%)=Z%
2110 RETURN
2149 REM --- (5) SUBROUTINE (MR%,A$,NX%,RC%) --- READ RECORD BY KEY
2150 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2155
2152 GET 2,PT%(K%) ' .. INPUT RECORD
2153 NX%=K%
2155 RETURN
2199 REM --- (6) SUBROUTINE (MR%,NX%,RC%) --- READ RECORD BY SEQUENCE
2200 IF NX%<0 OR NX%>PT%(MR%) THEN RC%=1: GOTO 2205
2203 GET 2, PT%(NX%)
2205 RETURN
2249 REM --- (7) SUBROUTINE (MR%) --- RESTORE POINTERS
2250 K%=0: Z%=INT((PT%(MR%)-1)/64)+1
2252 FOR J%=1 TO Z%
2253 FOR JJ%=1 TO 64: K%=K%+1:LSET PT$(JJ%)=MKI$(PT%(K%)): NEXT JJ%: PUT 1,J%
2254 NEXT J%
2255 K%=INT(MR%/64): IF Z%=K% THEN 2259
2257 K%=(K%-1)*64: FOR J%=1 TO 64: LSET PT$(J%)=MKI$(PT%(J%+K%)):NEXT J%:PUT
1,INT(MR%/64)
2259 RETURN
2279 REM --- (8) SUBROUTINE -- DISPLAY FILE STATISTICS
2280 PRINT " ":IF PT%(MR%)=0 THEN PRINT "*** NO RECORDS IN FILE": GOTO 2290
2282 PRINT " ** FILE STATISTICS **": PRINT " "
2283 PRINT " 1. RECORDS IN FILE: ";PT%(MR%)
2284 PRINT " 2. DELETED RECORDS: ";PT%(MR%-1)
2285 PRINT " 3. LOWEST KEY: ";KE$(PT%(1))
2286 PRINT " 4. HIGHEST KEY: ";KE$(PT%(PT%(MR%)))
2287 PRINT " "
2290 RETURN
2498 '
2499 REM --- SUBROUTINE (MR%,A$, K%) -- BINARY SEARCH
2500 IF PT%(MR%)=0 THEN K%=-1: RETURN
2502 LO%=0: HI%=PT%(MR%)+1
2504 M%=INT((LO%+HI%)/2)
2505 IF A$=KE$(PT%(M%)) THEN K%=M%: GOTO 2510
2506 IF A$>KE$(PT%(M%)) THEN LO%=M%: ELSE HI%=M%
2508 IF LO%+1 <> HI% THEN 2504 ELSE K%=-HI%
2510 RETURN
2518 '
2519 REM -- SUBROUTINE (MR%,PT%,Z%) -- LOCATE FREE RECORD IN DATA FILE
2520 IF PT%(MR%-1)=0 THEN Z%=PT%(MR%)+1: GOTO 2530
2522 J%=PT%(MR%):JJ%=PT%(MR%-1)
2524 Z%=PT%(MR%-1-JJ%): PT%(MR%-1)=PT%(MR%-1)-1: PT%(MR%-1-JJ%)=0
2530 RETURN
2538 '
2539 REM -- SUBROUTINE (MR%,K%,Z%) -- INSERT POINTER INTO POINTER VECTOR
2540 IF K%=PT%(MR%)+1 THEN 2548

```



```

2542 FOR J%=PT%(MR%)+1 TO K%+1 STEP -1
2544 PT%(J%)=PT%(J%-1)
2545 NEXT J%
2548 PT%(K%)=Z%: PT%(MR%)=PT%(MR%)+1
2550 RETURN
2997 '-----
2998 '          PROGRAM TO INITIALIZE INDEX FILE          -
2999 '-----
3000 PRINT " ":PRINT TAB(5); "*** INITIALIZE INDEX FILE ***":PRINT " "
3001 INPUT "> DRIVE TO CONTAIN DATA":UA$
3002 INPUT "> FILE NAME":F$
3004 INPUT "> MAXIMUM NUMBER OF RECORDS FILE WILL HOLD":MX%
3006 MR%=(INT((MX%+2)/64)+1)*64
3008 DIM PT$(64)
3009 '----- OPEN FILE AND SET POINTERS TO 0
3010 OPEN "R",#1,UA$+": "+F$,128
3012 FOR J%=1 TO 64: FIELD #1,(J%-1)*2 AS DU$,2 AS PT$(J%):NEXT J%
3014 ZR$=MKI$(0): FOR J%=1 TO 64: LSET PT$(J%)=ZR$: NEXT J%
3015 '----- STORE BLOCKS OF ZERO POINTERS
3016 FOR J%=1 TO MR%/64
3018 PUT 1,J%
3020 NEXT J%
3022 PRINT " ": PRINT "    INITIALIZATION COMPLETE ON DRIVE":UA$
3025 END

```

skam1.bas

"Keyed File Access in BASIC," by Stephen C. Perry.
 september, page 137. Also see skam.bas.

```

1 '-----
2 '          SAMPLE PROGRAM USING KEYED ACCESS ROUTINES          -
3 '-----
5 UA$="A" ' .. DRIVE CONTAINING DATA
16 OPEN "R",#2,UA$+":DATA.EMP",84 ' .. OPEN DATA FILE
17 FIELD #2, 9 AS KY$, 20 AS NM$, 6 AS BD$, 1 AS SX$, 3 AS JC$,
   20 AS A1$, 20 AS A2$, 5 AS ZP$
18 '
19 '   KY$ - ZIP CODE (KEY)  JC$ - JOB CODE
20 '   NM$ - NAME           A1$ - STREET ADDR.
21 '   BD$ - BIRTH DATE     A2$ - CITY-STATE
22 '   SX$ - SEX            ZP$ - ZIP CODE
23 '
25 MX%=150: F1$="PTR.EMP" ' ..INDEX FILE NAME
30 I1%=1: GOSUB 2000 ' ..INITIALIZE DATA STRUCTURE
31 '
32 INPUT "OPERATION (D,A,L,S,LA,U,Q)":Q$
33 IF Q$="D" THEN GOSUB 150: GOTO 32 ' DELETE
34 IF Q$="L" THEN GOSUB 180:
   GOTO 32 ' LIST INDIVIDUAL DATA
35 IF Q$="A" THEN GOSUB 100: GOTO 32 ' ADD
36 IF Q$="S" THEN I1%=8: GOSUB 2000:
   GOTO 32 ' DISPLAY STATISTICS
37 IF Q$="LA" THEN GOSUB 200:
   GOTO 32 ' LIST ALL RECORDS
38 IF Q$="U" THEN GOSUB 250: GOTO 32 ' UPDATE RECORD
40 IF Q$<>"Q" THEN 32
50 CLOSE: END
97 '
98 ' ***** ADD RECORD
99 '
100 INPUT "SS#":A$ : IF A$="END" THEN 120 ELSE IF
   LEN(A$)<>9 THEN 100
101 I1%=5:GOSUB 2000: IF RC$<>0 THEN LSET KY$=A$: GOTO 102 ELSE
   PRINT "*** ERROR - KEY ALREADY EXISTS": GOTO 100
102 INPUT "NAME":F$: LSET NM$=F$
105 INPUT "BIRTH DATE":F$: LSET BD$=F$
107 INPUT "SEX":F$: LSET SX$=F$
109 INPUT "JOB CODE":F$: LSET JC$=F$
110 INPUT "STREET":F$: LSET A1$=F$
111 INPUT "CITY-STATE":F$: LSET A2$=F$
112 INPUT "ZIP CODE":F$: LSET ZP$=F$
115 I1%=2: GOSUB 2000 ' .. ADD RECORD

```

(continued)

```

116 IF RC%=0 THEN 100 ELSE PRINT "** ERROR - RECORD CANNOT
    BE STORED": GOTO 100
120 II%=7: GOSUB 2000      '.. STORE POINTERS
122 RETURN
147 '
148 ' ***** DELETE RECORD
149 '
150 ST%=0
151 INPUT "CODE TO DELETE";A$: IF A$="END" THEN 156
152 II%=4: GOSUB 2000
154 IF RC%=0 THEN ST%=1 ELSE PRINT "** ERROR - KEY DOES NOT
    EXIST"
155 GOTO 151
156 IF ST%=1 THEN II%=7: GOSUB 2000      ' RESTORE POINTERS
    IF RECORD DELETED
158 RETURN
177 '
178 ' ***** LIST INDIVIDUAL RECORD
179 '
180 INPUT "SOCIAL SECURITY NUMBER";A$: IF A$="END" THEN 190
182 II%=5: GOSUB 2000: IF RC%<>0 THEN PRINT "**ERROR - KEY
    DOES NOT EXIST": GOTO 180
183 PRINT " "
184 PRINT "      NAME: ";NM$
185 PRINT "      JOB CODE: ";JC$
186 PRINT "BIRTH DATE: ";LEFT$(BD$,2);"/";MID$(BD$,3,2);
    "/";RIGHT$(BD$,2)
187 PRINT "      ADDRESS: ";A1$
188 PRINT TAB(13);A2$:PRINT ""
189 GOTO 180
190 RETURN
197 '
198 ' ***** LIST RANGE OF RECORDS
199 '
200 NX%=0: II%=6: K%=0
202 NX%=NX%+1: GOSUB 2000
204 IF RC%<>0 THEN 210
205 PRINT KY$,NM$
206 K%=K%+1: IF K%<10 THEN 202 ELSE INPUT ">";Q$      '.. PAUSE
207 IF Q$<>"END" THEN K%=0: GOTO 202
210 RETURN
247 '
248 ' ***** UPDATE RECORD
249 '
250 INPUT "SS#";A$: IF A$="END" THEN 270
252 II%=5:GOSUB 2000      '.. FETCH RECORD TO BE UPDATED
254 IF RC%=1 THEN PRINT "** ERROR - RECORD DOES NOT EXIST":
    GOTO 250
255 PRINT "NAME: /";NM$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET NM$=F$
257 PRINT "BIRTH DATE: /";BD$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET BD$=F$
258 PRINT "SEX: /";SX$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET SX$=F$
260 PRINT "JOB CODE: /";JC$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET JC$=F$
262 PRINT "STREET: /";A1$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET A1$=F$
263 PRINT "CITY-STATE: /";A2$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET A2$=F$
265 PRINT "ZIP CODE: /";ZP$;"/";: INPUT F$: IF LEN(F%)<>0
    THEN LSET ZP$=F$
266 II%=3: GOSUB 2000      '.. RESTORE UPDATED RECORD
268 PRINT " ": GOTO 250
270 RETURN
1995 '
1996 ' -----
1997 ' -                FILE MANAGEMENT SUBROUTINES
    (II%,MX%,F1$,A$,PT%,PT$, NX%,RC%) -
    -----
1998 ' -----
1999 '
2000 RC%=0: IF II%<1 OR II%>8 THEN RC%=1: RETURN
2001 IF II%=1 THEN 2006:      ' ELSE STORE VARIABLES
    USED BY SUBROUTINES
2004 ZZ%(1)=J$: ZZ%(2)=JJ$: ZZ%(3)=K$:ZZ%(4)=L$:
    ZZ%(5)=HI$: ZZ%(6)=Z%
2005 '

```



```

2006 ON II% GOSUB 2035,2080,2090,2100,2150,2200,2250,2280
2007 '
2008 IF II%=1 THEN 2010: ' ELSE RESTORE VARIABLES
      USED BY SUBROUTINES
2009 J%=ZZ%(1): JJ%=ZZ%(2): K%=ZZ%(3): LO%=ZZ%(4): HI%=ZZ%(5):
      Z%=ZZ%(6)
2010 RETURN
2034 REM --- (1) SUBROUTINE (MX%,F1$) --- INPUT POINTERS
      AND KEYS
2035 IF MX%<1 THEN RC%=1: RETURN
2037 MR%=(INT((MX%+2)/64)+1)*64
2038 DIM PT$(64),PT%(MR%),KE$(MX%),ZZ%(8)
2040 OPEN "R",#1,UA$+":"+F1$,128 ' INDEX FILE
2042 FOR J%=1 TO 64: FIELD #1,(J%-1)*2 AS DU$,
      2 AS PT$(J%): NEXT J%
2050 K%=0: IF LOF(1)=0 THEN 2062
2051 FOR J%=1 TO INT(MR%/64)
2052 GET 1,J% ' .. INPUT RECORD CONTAINING 64 POINTERS
2054 FOR JJ%=1 TO 64: K%=K%+1: PT%(K%)=CUI(PT$(JJ%)):
      NEXT JJ%
2055 NEXT J%
2056 '
2057 IF PT%(MR%)=0 THEN 2062
2058 FOR J%=1 TO PT%(MR%)+PT%(MR%-1)
2059 GET 2, J%: KE$(J%)=KY$
2060 NEXT J%
2062 RETURN
2079 REM --- (2) SUBROUTINE (MR%,A$, RC%) -- ADD
      RECORD TO FILE
2080 GOSUB 2500: IF K%>0 THEN RC%=1: GOTO 2088
2083 GOSUB 2520: IF Z%>MR%-1 THEN RC%=2: GOTO 2088
2085 K%=-K%:GOSUB 2540 ' .. INSERT POINTER . PT%(K%)=Z%
2086 KE$(Z%)=A$
2087 PUT 2,Z% ' .. STORE NEW RECORD
2088 RETURN
2089 REM -- (3) SUBROUTINE --- REWRITE RECORD
2090 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2098
2092 PUT 2,PT%(K%) ' .. STORE RECORD
2098 RETURN
2099 REM --- (4) SUBROUTINE (MR%,A$,RC%) --- DELETE
      A RECORD
2100 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2110
2102 Z%=PT%(K%): IF K%=PT%(MR%) THEN 2107
2104 FOR J%=K% TO PT%(MR%)-1: PT%(J%)=PT%(J%+1): NEXT J%
2107 JJ%=PT%(MR%-1)
2108 PT%(PT%(MR%))=0: PT%(MR%)=PT%(MR%)-1:
      PT%(MR%-1)=JJ%+1:PT%(MR%-2-JJ%)=Z%
2110 RETURN
2149 REM --- (5) SUBROUTINE (MR%,A$,NX%,RC%) --- READ
      RECORD BY KEY
2150 GOSUB 2500: IF K%<0 THEN RC%=1: GOTO 2155
2152 GET 2,PT%(K%) ' .. INPUT RECORD
2153 NX%=K%
2155 RETURN
2199 REM --- (6) SUBROUTINE (MR%,NX%,RC%) --- READ
      RECORD BY SEQUENCE
2200 IF NX%<0 OR NX%>PT%(MR%) THEN RC%=1: GOTO 2205
2203 GET 2, PT%(NX%)
2205 RETURN
2249 REM --- (7) SUBROUTINE (MR%) --- RESTORE POINTERS
2250 K%=0: Z%=INT((PT%(MR%)-1)/64)+1
2252 FOR J%=1 TO Z%
2253 FOR JJ%=1 TO 64: K%=K%+1:LSET PT$(JJ%)=MKI$(PT%(K%)):
      NEXT JJ%: PUT 1,J%
2254 NEXT J%
2255 K%=INT(MR%/64): IF Z%=K% THEN 2259
2257 K%=(K%-1)*64: FOR J%=1 TO 64:
      LSET PT$(J%)=MKI$(PT%(J%+K%)):
      NEXT J%:PUT 1,INT(MR%/64)
2259 RETURN
2279 REM --- (8) SUBROUTINE -- DISPLAY FILE STATISTICS
2280 PRINT " ":IF PT%(MR%)=0 THEN PRINT "*** NO RECORDS
      IN FILE": GOTO 2290
2282 PRINT " ** FILE STATISTICS **": PRINT " "
2283 PRINT " 1. RECORDS IN FILE: ";PT%(MR%)
2284 PRINT " 2. DELETED RECORDS: ";PT%(MR%-1)

```

(continued)

```

2285 PRINT " 3. LOWEST KEY: ";KE$(PT%(1))
2286 PRINT " 4. HIGHEST KEY: ";KE$(PT%(PT%(MR%)))
2287 PRINT " "
2290 RETURN
2498 '
2499 REM --- SUBROUTINE (MR%,A$, K%) -- BINARY SEARCH
2500 IF PT%(MR%)=0 THEN K%=-1: RETURN
2502 LO%=0: HI%=PT%(MR%)+1
2504 M%=INT((LO%+HI%)/2)
2505 IF A$=KE$(PT%(M%)) THEN K%=M%: GOTO 2510
2506 IF A$>KE$(PT%(M%)) THEN LO%=M%: ELSE HI%=M%
2508 IF LO%+1 <> HI% THEN 2504 ELSE K%=-HI%
2510 RETURN
2518 '
2519 REM -- SUBROUTINE (MR%,PT%,Z%) -- LOCATE FREE
RECORD IN DATA FILE
2520 IF PT%(MR%-1)=0 THEN Z%=PT%(MR%)+1: GOTO 2530
2522 J%=PT%(MR%):JJ%=PT%(MR%-1)
2524 Z%=PT%(MR%-1-JJ%): PT%(MR%-1)=PT%(MR%-1)-1:
PT%(MR%-1-JJ%)=0
2530 RETURN
2538 '
2539 REM -- SUBROUTINE (MR%,K%,Z%) -- INSERT POINTER
INTO POINTER VECTOR
2540 IF K%=PT%(MR%)+1 THEN 2548
2542 FOR J%=PT%(MR%)+1 TO K%+1 STEP -1
2544 PT%(J%)=PT%(J%-1)
2545 NEXT J%
2548 PT%(K%)=Z%: PT%(MR%)=PT%(MR%)+1
2550 RETURN
2997 ' -----
2998 ' - PROGRAM TO INITIALIZE INDEX FILE
2999 ' -----
3000 PRINT " ":PRINT TAB(5);"*** INITIALIZE INDEX
FILE ***":PRINT " "
3001 INPUT "> DRIVE TO CONTAIN DATA";UA$
3002 INPUT "> FILE NAME";F$
3004 INPUT "> MAXIMUM NUMBER OF RECORDS FILE WILL HOLD";MX%
3006 MR%=(INT((MX%+2)/64)+1)*64
3008 DIM PT$(64)
3009 '----- OPEN FILE AND SET
POINTERS TO 0
3010 OPEN "R",#1,UA$+":"+F$,128
3012 FOR J%=1 TO 64: FIELD #1,(J%-1)*2 AS DU$,2
AS PT$(J%):NEXT J%
3014 ZR$=MKI$(0): FOR J%=1 TO 64: LSET PT$(J%)=ZR$: NEXT J%
3015 '----- STORE BLOCKS OF
ZERO POINTERS
3016 FOR J%=1 TO MR%/64
3018 PUT 1,J%
3020 NEXT J%
3022 PRINT " ": PRINT " INITIALIZATION COMPLETE
ON DRIVE";UA$
3025 END

```

tbprolog.tst

"Turbo Prolog," by Namir Clement Shammass. September,
page 293.

Listing 1. Turbo Prolog List Reversal Test Program

/* Turbo Prolog List Reversal Test Program */

```

domains
    list = integer*

predicates
    append(list,list,list)
    writestring(list)
    lips(list)
    lipshort(list)
    rev(list,list)
    cycle(integer)

```



```

goal
    time(0,0,0,0),
    write("Enter cycle length "),
    readint(N),
    cycle(N),
    time(H,M,S,F),
    write("Time = ",H,":",M,":",S, ".",F),nl.

clauses
    append( [], L, L ).
    append( [Z|L1], L2, [Z|L3] ) :- append( L1, L2, L3 ).

    writestring( [] ).
    writestring( [H|T] ) :- write( H ), writestring( T ).

    lips(L) :- rev( [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,
47,48,49,50], L ).

    lipshort(L) :- rev( [1,2,3,4,5,6,7,8,9,10], L ).

    rev( [], [] ).
    rev( [H|T], L ) :- rev( T,Z), append( Z, [H], L ).

    cycle(0).
    cycle(N) :- N1 = N - 1, lips(_), cycle(N1).

```

Listing 2. Turbo Prolog Floating Point Test Program

```
/* Turbo Prolog Floating Point Test Program */
```

```

predicates
    calc(real,real)
    cycle(integer, real,real)

goal
    time(0,0,0,0),
    A = 2.71828,
    B = 3.14159,
    cycle(5000,A,B),
    time(H,M,S,F),
    write("Time = ",H,":",M,":",S, ".",F),nl.

clauses
    calc(A,B) :-
        C = 1.0,
        C1 = C * A,
        C2 = C1 * B,
        C3 = C2 / A,
        C = C3 / B,
        BOUND(C).

    cycle(0,A,B) :-
        C = 1.0,
        C1 = C * A,
        C2 = C1 * B,
        C3 = C2 / A,
        C = C3 / B,
        write("C = ",C),nl.

    cycle(N,A,B) :-
        calc(A,B),
        N1 = N - 1,
        cycle(N1,A,B).

```

(continued)

Listing 3. Turbo Prolog Sieve Test Program.

```

/* Turbo Prolog Sieve test program */

domains
    list = integer*

predicates
    integers(integer, integer, list)
    primes(integer, list)
    sift(list, list)
    remove(integer, list, list)
    cycle(integer)

goal
    time(0,0,0,0),
    cycle(10),
    time(H,M,S,F),
    write("Time = ",H,":",M,":",S,".",F),nl.

clauses

primes( Limit, Ps ) :-
    integers( 2, Limit, Is ),
    sift( Is, Ps ).

integers( Low, High, [Low|Rest] ) :-
    Low <= High, !, M = Low + 1,
    integers(M, High, Rest ).
integers( _,_,[] ).

sift( [], [] ).
sift( [I|Is], [I|Ps] ) :-
    remove(I,Is,New),
    sift( New, Ps ).

remove( _,[],[] ).
remove(P,[I|Is],[I|Nis] ) :-
    not( 0 = I mod P ),!,
    remove(P, Is, Nis).
remove(P,[I|Is],Nis) :-
    0 = I mod P,
    remove(P, Is, Nis).

cycle(0).
cycle(N) :-
    N1 = N - 1,
    primes(100,_),
    cycle(N1).

```

Listing 4. Turbo Prolog Math Functions Test Program.

```

/* Turbo Prolog Math Functions Test Program */

predicates
    cyclesqrt(integer, real)
    cycleln(integer, real)
    cycleexp(integer, real)
    cycleatan(integer, real)
    cyclesin(integer, real)

goal
    time(0,0,0,0),
    cyclesqrt(1000,_),
    time(0,0,0,0),
    cycleln(1000,_),
    time(0,0,0,0),
    cycleexp(1000,_),
    time(0,0,0,0),
    cycleatan(1000,_),
    time(0,0,0,0),
    cyclesin(1000,_).

```



```

clauses
cyclesqrt(0,R) :- R = sqrt(100.0),
                  time(H,M,S,F),
                  write("SQRT : ",H,":",M,":",S,".",F),nl,!.

cyclesqrt(N, R) :-
  N > 0, N1 = N - 1, R = sqrt(100.0), cyclesqrt(N1,R).

cycleln(0,R) :- R = ln(100.0),
                time(H,M,S,F),
                write("LN : ",H,":",M,":",S,".",F),nl.

cycleln(N, R) :-
  N > 0, N1 = N - 1, R = ln(100.0), cycleln(N1,R).

cycleexp(0,R) :- R = exp(10.0),
                 time(H,M,S,F),
                 write("EXP : ",H,":",M,":",S,".",F),nl.

cycleexp(N, R) :-
  N > 0, N1 = N - 1, R = exp(10.0), cycleexp(N1,R).

cycleatan(0,R) :- R = atan(10.0),
                  time(H,M,S,F),
                  write("ATAN : ",H,":",M,":",S,".",F),nl.

cycleatan(N, R) :-
  N > 0, N1 = N - 1, R = atan(10.0), cycleatan(N1,R).

cyclesin(0,R) :- R = sin(10.0),
                 time(H,M,S,F),
                 write("SIN : ",H,":",M,":",S,".",F),nl.

cyclesin(N, R) :-
  N > 0, N1 = N - 1, R = sin(10.0), cyclesin(N1,R).

```

Listing 5. Turbo Prolog Factorial Test Program.

```

/* Turbo Prolog Factorial Benchmark Test */

predicates
factorial(real,real)
repeat(integer,real)

goal
clearwindow,
write("Enter number of iterations "),
readint(Iter),nl,
write("Enter factorial number "),
readreal(Num),nl,nl,
time(0,0,0,0),
repeat(Iter,Num),
time(H,M,S,F),nl,
write("Time = ",H,":",M,":",S,".",F),nl.

clauses
factorial(1,1) :- !.
factorial(N,Result) :-
  N1 = N - 1,
  factorial(N1, Temporary),
  Result = N * Temporary.

repeat(0,R) :- factorial(R,X),
               write(X),nl.
repeat(N,R) :-
  factorial(R,_),
  N1 = N - 1,
  repeat(N1,R).

```

(continued)

Listing 6. Turbo Prolog Tower of Hanoi Test Program.

```
/* Turbo Prolog Tower of Hanoi Test Program */
```

```
domains
    list = integer*

predicates
    hanoi(integer)
    move(integer,symbol,symbol,symbol)
    move2(symbol,symbol,symbol)

goal
    write("Enter tower height "),
    readint(High),
    time(0,0,0,0),
    hanoi(High),
    time(H,M,S,F),
    write("Time : ",H,":",M,":",S,".",F),nl.
```

```
clauses
```

```
hanoi(N) :- move(N, left, center, right).
move(0, _, _, _) :- !.
move(N, A, B, C) :-
    M = N - 1,
    move(M, A, C, B),
    move2(bottom, A, B),
    move(M, C, B, A).
move2(bottom, A, B) :-
    write("Move the disk on "),
    write(A),
    write(" to "),
    write(B),
    nl.
```

Listing 7. Turbo Prolog Disk Write Test Program.

```
/* Turbo Prolog Disk write benchmark */
```

```
domains
    file = textfile

predicates
    send_text(integer)

goal
    openwrite(textfile,"a:tempo.dat"),
    writedevic(textfile),
    time(0,0,0,0),
    send_text(512),
    time(H,M,S,F),
    closeFile(textfile),
    writedevic(screen),
    write("Time = ",H,":",M,":",S,".",F), nl,
    write("DONE"),nl.
```

```
clauses
```

```
    send_text(0).
    send_text(N) :-
write("12345678123456781234567812345670123456781234567812345678123456701234567
81234567812345678123456781234567012345678123456781234567812345670"),
    nl, N1 = N - 1,
    send_text(N1).
```

Listing 8. Turbo Prolog Disk Read Program.

```
/* Turbo Prolog Disk Write Program */
```



```
domains
    file = textfile

predicates
    get_text(integer)

goal
    openread(textfile,"a:tempo.dat"),
    readdevice(textfile),
    time(0,0,0,0),
    get_text(512),
    time(H,M,S,F),
    closeFile(textfile),
    write("Time = ",H," ":"M"," ":"S"," ":"F), nl,
    write("DONE"),nl.
```

clauses

```
get_text(0).
get_text(N) :-
    ReadIn(Str),
    not (isname(Str)),
    N1 = N - 1,
    get_text(N1).
```

xmodem.asm

Programming Project: "Calculating CRCs by Bits and Bytes,"
by Greg Morse. September, page 114. Also see xmodem.c,
sdlc.asm, and ccitt.c.

```
* ... omitted lines same as for SDLC version .....  
*  
* Note changed order of these next four statements  
*  
Temp      rmb    1 ;;keep  
CRCReg     equ   . ;;       these  
CRCHI      rmb    1 ;;          statements  
CRL0       rmb    1 ;;              together  
stakbot    rmb   200-.  
CRCMem     equ   .  
*  
*.... omitted calling code same as for SDLC version ....  
*  
* : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : ~~~~~~  
* part B: Calculation subroutine  
*        - 28 AUG 85  
* PUBLIC DOMAIN SOFTWARE DONATED BY:  
* GREG MORSE Richmond B.C. CANADA  
*  
CRCRtn     equ    * begin BYTE-wise XMODEM-CRC  
*;+  
* Calculate the XMODEM-CRC value for a block of data  
* Polynomial used =  $X^{**}16 + X^{**}12 + X^{**}5 + 1$   
* on entry:  
* X points to data buffer  
* D contains number of chars in buffer <= 32767  
* mem locations Temp, CRCHI, CRL0 are don't care  
* They must be adjacent locations with Temp at the low address.  
* On Exit  
* X point past last char in buffer  
* D new CRC value for block  
* CRCHI,CRL0 new CRC for block  
* Temp destroyed  
*  
* DOC NOTES:  
* T = Data eor CRCHI  
* U = (T7 T6 T5 T4 0 0 0 0) eor (T3 T2 T1 T0 0 0 0 0)  
* During calcs CrchI not needed so is used as scratch area.  
* The inner loop from CRC.10 takes 90 cycles
```

(continued)

```

* The routine requires 1 byte of scratch
* 2 bytes for the result, and 6 bytes of stack
*;-
pshs y      save users y
leay d,x    start addr + byte count equals
pshs y      ending address plus 1
ldd #0      init CRC to all 0's
std CRCHReg initialize CRC area
oByt equ *  alt entry point for 1 byte CRC calc
* needs modified return see CRC.99
lda CRCHI
CRC.10 equ * Begin Inner Loop one loop per byte CYCLES
eor a,x+    A=T7 T6 T5 T4 T3 T2 T1 T0 (4)
tfr a,b     A=T7 T6 T5 T4 T3 T2 T1 T0 (11)
andb #$F0   B=T7 T6 T5 T4 0 0 0 0 (13)
anda #$0F    A= 0 0 0 0 T3 T2 T1 T0 (15)
std Temp    save A->temp; B->CRCHI (20)
lsrb        B= 0 T7 T6 T5 T4 0 0 0
lsrb        B= 0 0 T7 T6 T5 T4 0 0
lsrb        B= 0 0 0 T7 T6 T5 T4 0
lsrb        B= 0 0 0 0 T7 T6 T5 T4 (28)
eorb Temp   B= 0 0 0 0 U7 U6 U5 U4 (32)
eorb CRCHI  B= T7 T6 T5 T4 U7 U6 U5 U4 (36)
stb Temp    temp= T7 T6 T5 T4 U7 U6 U5 U4 (40)
clra        (42)
lslb        A= 0 0 0 0 0 0 0 T7 (46)
rola        A= 0 0 0 0 0 0 0 T7 T6 (50)
lslb        A= 0 0 0 0 0 0 T7 T6 T5 (54)
rola        B= U7 U6 U5 U4 0 0 0 0
lslb        A= 0 0 0 0 T7 T6 T5 T4 (68)
rola        B= U6 U5 U4 0 0 0 0 0
stb CRCHI   A= 0 0 0 T7 T6 T5 T4 U7 (76)
lslb        (70)
rola        A= new CRCHI (74)
eorb CRCHI  B= new CRCLo (78)
eorb Temp   one data byte all done (83)
std CRCHReg
CRC.99 equ * make RTS if doing only 1 byte
cmpx ,      x past end of buffer? (87)
blo CRC.10  if not repeat inner loop Note: (90)
* A still = CRCHI
leas 2,s    pop topaddr
puls y,pc   restore y and return
*
FoxMsg equ *
fcc /THE,QUICK,BROWN,FOX,0123456789/
FoxSiz equ *-FoxMsg
USE shoregs.src
emod
CRCsiz equ * os9 CRC bytes

```

xmodem.c

Programming Project: "Calculating CRCs by Bits and Bytes,"
by Greg Morse. September, page 114. Also see ccitt.c,
sdlc.asm, and xmodem.asm.

```

/* Sample bit-oriented CRC routine */
/* Adapted from YMODEM protocol reference */

/* Calculate CRC on a block of data. */
/* Ptr points to block of characters, */
/* count gives size of buffer. */
/* This program returns the CRC with the LSB */
/* of the CRC in the high bit of the */
/* result integer. */
/* XMODEM deviates from the CCITT standard in */
/* that it does not use */
/* the LSB of the data first, nor does it */
/* initialize the CRC to all */

```



```
/* ones as specified by the standard. */
```

```
int calcrc(ptr, count)
char *ptr;
int count;
{
    unsigned int crc;
    int i;
    crc = 0; /* note not 0xFFFF */
    while (--count >= 0) {
        i = (int) *ptr++; /* convert data char to int */
        i = i << 8; /* shift char to high byte */
        crc = crc ^ i; /* add current data to current */
        /* remainder modifies only least */
        /* sig 8 bits (high byte) of CRC */
        for (i=0; i<8; ++i) /* loop for each bit */
            if (crc & 0x8000) { /* test D XOR R0 */
                crc = (crc << 1); /* discard LSB of CRC and */
                                /* append zero */
                crc = crc ^ 0x1021; /* XOR with low 16 bits */
                                /* of CCITT polynomial */
                                /* because CRC is stored LSB 1st */
                                /* polynomial written MSB first */
            } /* endif */
        else
            crc = crc << 1; /* discard LSB & append 0 */
    } /* end while */
    return (crc & 0xFFFF); /*
    /* 16-bit result for whole block */
    } /* end calcrc */
```



D·I·S·K·S A·N·D D·O·W·N·L·O·A·D·S

ORDERING DISKS OF BYTE LISTINGS

Listings that accompany BYTE articles are available in a variety of disk formats and on Causer Softstrip. Each disk package (which sometimes consists of more than one disk) contains an entire month's listings. If you want to order a disk package from a previous month, please call (603) 924-9281 to find out how many disks it includes. To order listings (for noncommercial use only), fill out this form and send a check or money order in the correct amount to:

BYTE Listings
One Phoenix Mill Lane
Peterborough, NH 03458

All prices include postage. Program listings can also be downloaded via BYTEnet Listings at (617) 861-9764.

BYTE issue: _____

COMMON 5¼-INCH FORMATS

All cost \$8.95, \$10.95 outside U.S.A. Annual subscription is \$69.95, \$89.95 outside U.S.A.

- ☐ Apple II 5¼-inch DOS 3.3
- ☐ Apple II 5¼-inch ProDOS
- ☐ Hewlett-Packard 125
- ☐ IBM PC
- ☐ Kaypro 2 CP/M
- ☐ Texas Instruments Professional
- ☐ TRS-80 Model III
- ☐ TRS-80 Model 4
- ☐ Zenith Z-100

COMMON 3½-INCH FORMATS

All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

- ☐ Apple Macintosh
- ☐ Atari 520ST
- ☐ Commodore Amiga
- ☐ Data General/One
- ☐ Hewlett-Packard 150

CP/M STANDARD 8-INCH FORMAT

All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

- ☐ Single-sided single-density
- ☐ Double-sided double-density

OTHER FORMATS

Due to the diversity of requests and the custom work involved, there will be some delay in fulfilling these requests. All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

Size ☐ 8-inch ☐ 5¼-inch ☐ 3½-inch
Machine _____

SEND TO:

Name _____

Street _____

City _____ State or Province _____

Postal Code _____ Country _____

Check or money order enclosed for \$ _____

BULLETIN BOARDS IN CANADA

Listed below are some computer bulletin boards that carry program listings from BYTE. Programs are for noncommercial use in connection with BYTE articles only. Some BBSs may charge an annual maintenance fee, and you must pay your own telephone charges.

Western Canadian Distribution Center (3420 48th St., Edmonton, Alberta T6L 3R5) will be supplying listings to its member bulletin board systems.

Edmonton, Alberta, (403) 454-6093

Meadowlark, Alberta, (403) 435-6579

Montreal, Quebec, PComm Systems, (514) 989-9450

Prince George, British Columbia, (604) 562-9519

Regina, Saskatchewan, (306) 586-5585

Canadian Remote Systems, Toronto

Toronto, Ontario, Epson Club of Toronto (EPCOT), (416) 635-9600

Winnipeg, Manitoba, (204) 452-5529

In addition, arrangements for BYTEnet Listings have been made with one or more system operators in the following nations: Australia, Austria, Brazil, Denmark, France, Hong Kong, Indonesia, Italy, Japan, Malaysia, The Netherlands, Nigeria, Norway, Saudi Arabia, Singapore, Sweden, Switzerland, United Kingdom, and West Germany. Contact us at (603) 924-9281 for an up-to-date list. ■

EDITORIAL CALENDAR

1987

- MAY — DESKTOP PUBLISHING:** An exploration of the hardware and software needed for desktop publishing, from page description languages to high-resolution printers and typesetting back ends.
- JUNE — COMPUTER-AIDED DESIGN:** The anatomy of computer-aided design/drafting software, the graphics display devices needed for CAD, and the data structures used by CAD programs to export data to other applications.
- JULY — LOCAL AREA NETWORKS:** The technology of linking personal computers together to share data files, programs, and peripheral devices.
- AUGUST — PROLOG:** A look at logic programming with articles on tips and techniques and explorations of the tasks Prolog is best suited for.
- SEPTEMBER — PRINTER TECHNOLOGIES:** An examination of the state of the art in printer technologies, including laser, liquid-crystal shutter, and ink-jet technologies.
- OCTOBER — HEURISTIC ALGORITHMS:** Artificial intelligence techniques for giving computers the ability to learn from experience.
- NOVEMBER — HIGH-PERFORMANCE WORKSTATIONS:** A tour of the technology underlying the workstations used by scientists and engineers in computer-aided engineering/design.
- DECEMBER — NATURAL LANGUAGE PROCESSING:** The technology of getting computers to understand the natural language of man.

1988

- JANUARY — MANAGING MEGABYTES:** Looking at the ways computers store and retrieve data in situations where disk space is measured in gigabytes and memory is measured in megabytes. Also a look at the new applications that mega-memory and storage will permit.
- FEBRUARY — LISP:** A BYTE reexamination of the original language of artificial intelligence research.
- MARCH — FLOATING-POINT PROCESSORS:** A look at the processors that speed the computation of mathematical operations in personal computers, including coprocessors and array processors.
- APRIL — MEMORY MANAGEMENT:** The hardware and software issues in managing a personal computer's memory space.
- MAY — CPU ARCHITECTURES:** An exploration of the latest 32-bit microprocessors, including digital signal processors and programmable graphics processors.

Six great reasons to join **BIX** today

- **Over 140 microcomputer-related conferences:**

Join only those subjects that interest you and change selections at any time. Take part when it's convenient for you. Share information, opinions and ideas in focused discussions with other BIX users who share your interests. Easy commands and conference digests help you quickly locate important information.

- **Monthly conference specials:**

BIX specials connect you with invited experts in leading-edge topics—CD-ROM, MIDI, OS-9 and more. They're all part of your BIX membership.

- **Microbytes daily:**

Get up-to-the-minute industry news and new product information by joining Microbytes Daily and What's New Hardware and Software.

- **Public domain software:**

Yours for the downloading, including programs from BYTE articles and a growing library of PD listings.

- **Electronic mail:**

Exchange private messages with BYTE editors and authors and other BIX users.

- **Vendor support:**

A growing number of microcomputer manufacturers use BIX to answer your questions about their products and how to use them for peak performance.

What BIX Costs...How You Pay

ONETIME REGISTRATION FEE: \$25

Hourly Charges: (Your Time of Access)	Off-Peak 6PM-7AM Weekdays Plus Weekends & Holidays	Peak 7AM-6PM Weekdays
BIX	\$9	\$12
Tymnet*	\$2	\$6
TOTAL	\$11/hr.	\$18/hr.**

* Continental U.S. BIX is accessible via Tymnet from throughout the U.S. at charges much less than regular long distance. Call the BIX helpline number listed below for the Tymnet number near you or Tymnet at 1-800-336-0149.

** User is billed for time on system (i.e., 1/2 Hr. Off-Peak w/Tymnet = \$5.50 charge.)

BIX and Tymnet charges billed by Visa or Mastercard only.

BIX HELPLINE

(8:30 AM-11:30 PM Eastern Weekdays)

U.S. (except NH)—1-800-227-BYTE

Elsewhere (603) 924-7681



We'll
Send
You a

BIX User's Manual and Subscriber Agreement
as Soon as We've Processed Your Registration.
**JOIN THE EXCITING WORLD
OF BIX TODAY!**

JOIN BIX RIGHT NOW:

Set your computer's telecommunications program for full duplex, 8-bit characters, even parity, 1 stop bit OR 7-bit characters, even parity, 1 stop using 300 or 1200 baud.

Call your local Tymnet number and respond as follows:

Tymnet Prompt

Garble or "terminal identifier"
login:
password:
mhis login:
BIX Logo—Name:

You Enter

a
byteneti <CR>
mgh <CR>
bix <CR>
new <CR>

After you register on-line, you're immediately taken to the BIX learn conference and can start using the system right away.

FOREIGN ACCESS:

To access BIX from foreign countries, you must have an account with your local Postal Telephone & Telegraph (PTT) company. From your PTT enter 310600157878. Then enter bix <CR> and new <CR> at the prompts. Call or write us for PTT contact information.

BIX

ONE PHOENIX MILL LANE
PETERBOROUGH, NH 03458
(603) 924-9281

Announcing BYTE's New Subscriber Benefits Program

Your BYTE subscription brings you a complete diet of the latest in microcomputer technology every 30 days. The kind of broad-based objective coverage you read in every issue. *In addition*, your subscription carries a wealth of other benefits. Check the check list:

DISCOUNTS

- ✓ 13 issues instead of 12 if you send payment with subscription order.
- ✓ One-year subscription at \$21 (50% off cover price).
- ✓ Two-year subscription at \$38.
- ✓ Three-year subscription at \$55.
- ✓ One-year GROUP subscription for ten or more at \$17.50 each. (Call or write for details.)

SERVICES

- ✓ **BIX:** BYTE's Information Exchange puts you on-line 24 hours a day with your peers via computer conferencing and electronic mail. All you need to sign up is a microcomputer, a modem, and telecomm software.
- ✓ **Reader Service:** For information on products advertised in BYTE, circle the numbers on the Reader Service card enclosed in each issue that correspond to the numbers for the advertisers you select. Drop it in the mail and we'll get your inquiries to the advertisers.
- ✓ **TIPS:** BYTE's Telephone Inquiry System is available to



subscribers who need *fast response*. After obtaining your Subscriber I.D. Card, dial TIPS and enter your inquiries. You'll save as much as ten days over the response to Reader Service cards.

- ✓ **Disks and Downloads:** Listings of programs that accompany BYTE articles are now available free on the BYTenet bulletin board, and on disk or in quarterly printed supplements.
- ✓ **Microform:** BYTE is available in microform from University Microfilm International in the U.S. and Europe.
- ✓ **BYTE's BOMB:** BYTE's Ongoing Monitor Box is your direct line to the editor's desk. Each month, you can rate the articles via the Reader Service card. Your feedback helps us

keep up to date on your information needs.

- ✓ **Customer Service:** If you have a problem with, or a question about, your subscription, you may phone us during regular business hours (Eastern time) at our toll-free number: 800-258-5485. You can also use Customer Service to obtain back issues and editorial indexes.

BONUSES

- ✓ **Annual Separate Issues:** In addition to BYTE's 12 monthly issues, subscribers also receive our annual IBM PC issue free of charge, as well as any other annual issues BYTE may produce.
- ✓ **BYTE Deck:** Subscribers receive five BYTE postcard deck mailings each year—a direct response system for you to obtain information on advertised products through return mail.

To be on the leading edge of microcomputer technology and receive all the aforementioned benefits, make a career decision today. Call toll-free weekdays, 8:30am to 4:30pm Eastern time: 800-258-5485.

*And... welcome to
BYTE country!*

BYTE
THE SMALL SYSTEMS JOURNAL

